HEATH'S ROBOT "HERO" 69 EXPERIMENTS: FUNDAMENTALS and APPLICATIONS

by
HOWARD BOYET

MICROPROCESSOR TRAINING INC.
14 East 8th Street
New York, N.Y. 10003
212-473/4947

HEATH'S ROBOT "HERO"

69 EXPERIMENTS:

FUNDAMENTALS AND APPLICATIONS

By

Howard Boyet
Microprocessor Training Inc.
14 East 8th Street, N.Y., N.Y. 10003
(212) 473-4947

Other books by the author published by Microprocessor Training, Inc.:

1. 8080 Microcomputer Experiments, H. Boyet, 2nd Ed, 1979, 396 pp.
2. Applications Experiments With The 8080/8085 Microprocessor-Microcontroller, H. Boyet and R. Katz, 1979, 751 pp, (2 Volumes).
3. The 8085 Microprocessor, Fundamentals and Applications - 84 Control Experiments on the Intel SDK-85, Vol. 1, H. Boyet, 2nd Ed, 452 pp, 1982.
4. The 8085/SDK-85, Vol. 2, 54 More Advanced Control Experiments, H. Boyet, 394 pp. 1981.
5. The 8051 Single Chip Microcomputer: Programming, Interfacing, Applications. 81 Experiments With The Intel SDK-51, H. Boyet and R. Katz, 396 pp. 1982.
6. The 16-Bit 8096 Single chip Microcomputer: 122 Experiments with Intel's iSBE-96 Emulator. H. Boyet & R. Katz, 638pp, 1986.

DEDICATION

To Migdalia DeLeon and Gigi Rebecca Boyet,
the inspirations behind this book.

Copyright © 1983 by Howard Boyet.

The information in this book has been checked and is believed to be entirely reliable. However, neither the author nor MTI Publications (Microprocessor Training Inc.) assumes responsibility for inaccuracies. The book is neither an endorsement of any manufacturer, any product, or any process. No responsibility is assumed for its use or infringements of patents or rights of others which may result in its use. No part of this book may be stored in a retrieval system, reproduced, transmitted, or copies in any way or form without prior written permission of the author and publisher. World rights reserved.

Printing 13, 1987

Library of Congress catalog card number: 83-62803

Printed in the United States of America

CONTENTS

CHAPTER 1: INTRODUCTION AND GENERAL INFORMATION. 1

General Remarks 1

Entering a Program on HERO 2

Running a Program 3

Examining and Modifying a Program 3

Single Step, Breakpoints 3

A Note on HERO's Memory Allocation 4

Theory of Operation of HERO 4

Appendices A and B in This Manual 4

Experiments With HERO's Monitor Subroutines 4

A. Expt. 1-1: Input a Character ("INCH") 5

B. Expt. 1-1: Output a HEX Number ("OUTHEX") 5

C. Expt. 1-2: Reset the Display ("REDIS") 5

D. Expt. 1-3: Output a Character("OUTCH") 6

E. Expt. 1-4: Output a String of Characters ("OUTSTR") 6

F. Expt. 1-5: Clear Display ("CLRDIS") 7

G. Expt. 1-6: Output a Byte ("OUTBYT") 7

H. Expt. 1-7: Create and Display a Byte From Two Keyboard Entries 7

I. Expt. 1-8: Input and Display HEX Byte From Keyboard ("IHB") 8

J. Expt. 1-9: Display Bytes ("DISPLAY") 9

CHAPTER 2: MACHINE LANGUAGE CONTROL OF HERO's I/Os. 11

Discussion 11

A. HERO's I/O Addresses and Their Descriptions 11

HERO's Output Ports 11

HERO's Inport Ports 12

B. Experiments Using HERO's I/O Ports 13

Expt. 2-1: The Experimental Board (Outport C220 and Inport C2A0) 14

Working With Outport C2E0 14

Expt. 2-2: Using Outport C2E0 to Enable Sound Sensor and Inport C240 to Read Sound Level 15

Expt. 2-3: Using Outport C2E0 and Inport C240 to Enable and Read Light Detector 16

Expt. 2-4: Using Outport C2E0 to Enable Display 17

Expt. 2-5: Using Outport C2E0 to Enable HERO's Motion Detector. Detecting Motion 17

Expt. 2-6: Using Outport C2E0 to Enable or Disable Motors (Main Power) ... 19

Expt. 2-7: Using Outports C2E0, C240, and C2C0 to Create Phoneme Sounds From the Speech Synthesizer 20

Expt. 2-8: Using Outport C2E0 to Enable or Disable the Sonar Board (Ultrasonic Ranging) 22

Expt. 2-9: Reading the Sonar Timer at Inport C220 23

Expt. 2-10: Using Inport C260 to Input the Status of the Limit Switches ... 23

Expt. 2-11: Polling the Limit Switches at Port C260 In a Robot Safety Application. Readout of Which Motors Reached Their Limits 25

Expt. 2-12: Using Outport C2A0 to Control Main Drive (the Base): Forward/Reverse, On/Off, Speed 28

Expt. 2-13: Reading Inport C280 for the Remote Teaching Pendant Settings (Codes) Including the Sleep/Normal Switch 29

Expt. 2-14: Using Outports C2C0,C280, and C260 to Gain Direct Machine Code Access to HERO's 7 steppers. Driving the Steppers at any Speed and Counting the Steps 33

CHAPTER 3: HERO'S USER-DEDICATED KEYS; SPECIAL RAM LOCATIONS; ITS INTERRUPT PORT; ITS DISPLAY PORTS; REAL TIME CLOCK PORTS. 39

Expt. 3-1: HERO's User-Dedicated Keys: 9,C,F 39

Expt. 3-2: Reading the 7 Stepper Motor Positions From RAM Buffer Locations 0000-0006 "On The Fly" 41

Expt. 3-3: Special RAM Location 0EE1 43

Expt. 3-4: User Polling of Interrupt Inport C200 For Source of Interrupts. Servicing Them 43

Expt. 3-5: HERO's Monitor Polls C200 Source of Interrupts. Jumps to Dedicated Locations for Servicing. Examples and Applications 47

Expt. 3-6: IRQ Interrupts and the Stack 49

Expt. 3-7: The 6 LED Display. I/O Outports C1YX 50

Expt. 3-8: HERO's Real Time Clock. Outports C2C0,C300 and Inport C300. Application 51

CHAPTER 4: THE INTERPRETER BEHIND A ROBOT LANGUAGE. WRITING YOUR OWN INTERPRETER FOR YOUR ROBOT COMMANDS. 55

Theory 55

Expt. 4-1: Writing Your Own Interpreter to Execute Your Own Robot Language Commands 56

CHAPTER 5: LIGHT, SOUND, AND SPEECH WITH HERO. APPLICATIONS. 63

Introduction 63

Expt. 5-1: Robot Polls for Light. If Detected it Speaks Till Light is Removed 64

Expt. 5-2: Robot Polls for Sound. If Detected it Speaks till Sound Source Removed 65

Expt. 5-3: Robot Responds Only When Both Light and Sound Exceed Critical Levels Simultaneously 66

Expt. 5-4: The "Sleep Command". Robot "Sleeps" and then Polls for Light. Application to "Guard Duty" 68

Expt. 5-5: Measurement and Display of the Time Duration of an Event. The Pause Command. Robot Applications 69

Expt. 5-6: When Dark, Robot Reaches Up to Turn Light Switch on. Repeats Every Time Light Level Recedes Below Critical 71

Expt. 5-7: Each Time Sound Level Goes Above Critical HERO Says "Turn The TV Down." 72

Expt. 5-8: A Conveyor Belt Problem: Filling Cartons Using Light to Sense if Carton Present or Not 73

Expt. 5-9: A Robot On Guard: "Friend or Foe" 75

Expt. 5-10: "Motor Move Continue" (CC or DC) vs. "Motor Move Wait" (C3 or D3) Commands 76

Expt. 5-11: Searching For Light. Head Scan Stops When Found, Resumes When Light Disappears 79

Expt. 5-12: A Robot Parts - Placing Operation In Manufacturing or Construction Guided by a Light Source 81

CHAPTER 6: MOTION DETECTION AND ULTRASONIC RANGING WITH HERO. APPLICATIONS. 85

Introduction 85

Expt. 6-1: Robot Speaks and Moves Forward Each Time Motion is Detected 86

Expt. 6-2: Count-Up and Display of Number of Times Motion is Detected. Applications 87

Expt. 6-3: A Robot Safety Application: Robot Drives Away From Nearby Motion and Only Drives Back to Resume Activity When a Key is Pressed 88

Expt. 6-4: Motion Detected (Arrival of a Part). Robot Moves Forward, Takes Part, Places it. Robot Drives Back to Await Another Part ... 90

Expt. 6-5: Robot Drives Forward Only When Nearest Target is Greater Than Critical Distance Away, Otherwise Stops. Resumes Motion if Target Recedes. Application 91

Expt. 6-6: Robot Stops Only if Target Distance is Less Than Critical or Light is Sensed 92

Expt. 6-7: Robot Scans for First Target Outside a Critical Radius ("Early Warning System") 93

Expt. 6-8: Robot Walks Along An Enclosure Looking for an Opening. Turns and Walks Through it If and When Found 95

CHAPTER 7: HERO'S REMOTE MANUAL CONTROL TEACHING PENDANT. THEORY BEHIND ITS OPERATION. APPLICATIONS. MODIFYING TAUGHT PROGRAMS. 97

Introduction 97

A. The Pendant in Remote Manual Mode (Tele-Operator) 98

Expt. 7-1: A Program Implementing Remote Manual Control Actions of Motors Corresponding to Those Required by Various Pendant Settings 99

B. The Pendant Used in the Teach or Learn Mode. Storing, Executing, and Modifying the Taught Program 102

Expt. 7-2: A Learned, Stored Action. Its Replay. Modifications From Keyboard 103

Expt. 7-3: Another Teach Pendant Application Example 105

Expt. 7-4: What's Behind the Teach Pendant. Writing Your Own Utility Program to do What "Utility Mode 7" in HERO's Monitor Does 106

CHAPTER 8: APPLICATIONS OF AN INDUSTRIAL OR MANUFACTURING NATURE (SOME FUN AND GAMES TOO). 109

Remarks 109

Expt. 8-1: A Generalized Program to Implement Teaching a Robot From the Keyboard. Replay Action Once or Finite Number of Times. Editing the Taught Program. Abort Option. The E3 Indexed Motor Move Command 109

Expt. 8-2: A "Pick and Place" Robot Taught From the Keyboard 115

Expt. 8-3: Choosing From a Menu of Robot Tasks Stored as Tables. The F3 (FC) Extended Motor Move Command 117

Expt. 8-4: A Two-Sensor Application: Filling Cartons With Parts As They Arrive on a Conveyor Belt. Carton Leaves When Filled 121

Expt. 8-5: Simulating "Camera-Vision" Detection of The Location of a Part: Part Picked, Treated, Placed on Piece; Piece Placed on Conveyor Belt. Repeat 123

Expt. 8-6: A Busy Robot Plays "Catch-Up" With a Moving Conveyor Belt to Place the Part 125

Expt. 8-7: Placing Parts From Two Bins on to an Assembly. Testing For Empty Bins (Finished Piece). Quality Control Test of the Finished Piece (Reject or Pass on). Count of Good/Bad Pieces Produced 128

Expt. 8-8: A Robot Learns From its Environment as it Does its Task. Reprograms Itself and Does Task Differently Next Time 133

Expt. 8-9: A Robot is Commanded to Sleep or Work, Then "Goes to Lunch", "Feeds Self", Comments on Food, Returns to Home Base to Resume Sleep or Work, etc. 135

Expt. 8-10: HERO Welcomes Participants at a Seminar in Robotics. Serves Them 138

Expt. 8-11: HERO Goes to a Dance 140

Expt. 8-12: A Robot Sorts Parts Coming Down a Conveyor Belt According to Size and Places Them in Appropriate Bins 142

Expt. 8-13: HERO Used in X-Y Plotting (Applications to Graphics, Machine Pattern Cutting, Spray Painting, Numeric Controls, etc.) 144

Expt. 8-14: Limit Switch Control of Robot Operations. Application 147

APPENDIX A: USEFUL HERO REFERENCE MATERIAL ON PROGRAMMING AND I/Os. THE 6800 INSTRUCTION SET. 6808 PIN-OUT 149

APPENDIX B: A TABLE OF SS MOTOR SELECT/SPEED/DIRECTION BYTES TO BE APPENDED TO THE C3,CC,D3,DC MOTOR MOVE COMMANDS. LIMITS ON ABSOLUTE POSITIONS XX FOR VARIOUS HERO MOTORS 161

PREFACE

The Heath* Co.'s robot "HERO" appeared around the turn of the year 1983. In the writer's opinion it is an outstanding and rather unique educational vehicle in the field of microcomputer control of robots and lends itself easily, as well, to use as a prototyper and experimenter.

HERO stresses many aspects of modern day robots that make it stand apart in the educational area: seven degrees of freedom plus lateral movement; light and sound sensors; motion detection and distance sensing capabilities; speech synthesis; a variety of I/Os (sensors, steppers, drive, interrupt lines, remote pendant control, teach pendant, limit switches, sonar, real time clock, speech synthesizer, enable/disable lines to various I/O boards, an input/output experimental board); and above all, microcomputer control of all the I/Os with all the power of flexibility and versatility that the latter affords. (HERO's microprocessor is Motorola's 6808).

We are not saying that HERO is an industrial robot - it is not nor was it intended to be (as of this writing). But you can learn robotics and microcomputer control of robots with it, and it can be used to simulate, prototype, and experiment with real-world industrial and manufacturing type applications. This manual of experiments uses it for those purposes.

Further, the robot in general will also have future application and use in non-industrial environments: security; chores in the home, hospital, and senior citizen institutions; possibly in the office, etc. - in short wherever it will be socially helpful to man and economically advantageous. Some of these areas are also addressed by the experiments in this manual.

And, importantly, the experiments worked out, prototyped, and tried on HERO can always be transferred to other possibly more appropriate robot systems that will be used in the actual industrial or non-industrial application.

Finally, in addition to the above, HERO justifies itself as an excellent vehicle for learning and working with microcomputer control of real-world I/Os in general.

This manual of 69 experiments with HERO was written in the spirit of the above remarks. We are after knowledge and understanding, with applications in the real world of robots. We want to be more than just users of robots. Our aim is to become robot innovators capable of conceiving and implementing imaginative programs to control and use robot I/Os in imaginative ways in a variety of areas - industrial and non-industrial. We stress programming and I/O fundamentals, and applications built around those fundamentals. It is our hope that this manual offers a solid bridge over a void in beginning robot education at this time - a bridge leading to the realization of the above aims.

The user of this manual should have previous exposure to microprocessors and micro-

*A subsidiary of Zenith Radio Co.

computers, preferably hands-on. We do not start "from scratch" as far as microprocessors are concerned.

A perusal of the table of contents will tell you better that we can here, the scope of the material, the experiments, their logical progression and their stress on the various aspects of the fundamentals and applications. Learning any new subject should be fun - that's an important ingredient of successful teaching. We think you'll have that fun as you learn and work your way through this manual.

The author wishes to express his deep appreciation for important help on the part of various individuals: Messrs. James Arleth and Henry Massalin, students in the engineering school of Cooper Union (N.Y., N.Y.) and Stephen Arleth for an excellent and efficient job of assembling HERO; Mr. Vincent M. Altamuro, president of Management Research Consultants (Toms River, N.J.) for suggesting the themes of several important industrial/manufacturing types of applications experiments in Chapter 8; all the participants from Western Electric Co., Inc. (Indianapolis) who attended two robot seminars presented there by Mr. Altamuro and the author, for their excellent suggestions pertaining to some of the material and experiments in this manual when those experiments were studied and tried as part of the course; Mr. Dave Donville and Barbara Starks of Western Electric Co.'s Engineering Personnel Development department in Indianapolis for their important cooperation; Messrs. G. Gladish and R. Rudolf of Western Electric, Indianapolis for contributing one of the experiments (8-11) in this manual; Messrs. Doug Bonham, Matt Cutter, James Lytle, Tom Haglund, and Doug Wise of the Heath Co. (Benton Harbor, MI) for their cooperation and permission to reproduce some of their material in Appendix A; and to Margi Scallo with whom it is always a great pleasure to work on the word processing of my books.

Howard Boyet
Microprocessor Training Inc.
October 1983
New York, New York

CHAPTER 1

INTRODUCTION AND GENERAL INFORMATION

The writer considers Heath Co.'s HERO a serious learning and prototyping tool in the field of robotics. The flexible microcomputer control of its I/Os representing its seven degrees of freedom, as well as lateral movement, is one of its strong points and is in keeping with the present day (and undoubtedly future) trend toward microcomputer controlled (smart) robots. We are not, however, saying that HERO is an industrial robot - it is not and was not intended to be. But you can have it simulate and prototype real-world industrial robot type applications. That we do here. It is an excellent educational robot.

This manual of experiments with HERO is written in the spirit of the above remarks.

The manual stresses two goals: The software and I/O fundamentals, and real world applications built around these fundamentals. We feel the realization of these goals in this manual will fill a serious educational void with respect to HERO in particular and robots in general. We want to be robot innovators, not just users.

Chapter 1 concerns itself with the following material: how to enter, run, examine, and modify a program with HERO; HERO's memory allocation; the difference between HERO's "Robot Language" (R.L.) HEX code commands and the 6800 microprocessor's machine language (M.L.) HEX Op Codes comprising its instruction set (and the attendant precautions that must be taken by the programmer when writing a program consisting of both R.L. and M.L. codes); and experiments demonstrating working with powerful and useful subroutines in HERO's monitor - subroutines that we shall have many occasions to use and which will add to the power of our robot experiments. They will help give them real world qualities in an industrial controls and applications setting.

While HERO uses Motorola's 6808 microprocessor as its controller, the instruction set is identical to that of the 6800 microprocessor. We shall always refer to the 6800 microprocessor when needing to do so. While certain pin assignments on the 6808 differ from those on the 6800, they do not concern us here. (see App. A).

From time to time we will be referring to the "Heath Robot Technical Manual" and to the "Heath Robot User's Guide" supplied by the Heath Co. to purchasers of HERO. References to them, while helpful, are in no way essential to the understanding and implementation of the experiments in this manual. Our manual is self contained.

We feel that all the experiments in this manual will be executable on any updated versions of HERO that Heath Co. may come out with in the future, although,

at this writing, we cannot be certain of that as Heath Co. has not yet released any information on possible forthcoming developments.

If you now consult Appendix A for the "Programmer's Information Sheet" (or p.115 Heath's Robot Technical Manual) you will notice 37 "Interpreter Commands" (in HEX) which constitute HERO's "Robot Language" (R.L.). These are not 6800 machine language Op Codes, as you can verify by perusing the 6800 instruction set (see Appendix A). They are Heath codes for desired robot actions of one kind or another. These command codes and corresponding robot actions are executed by the various machine language (6800) routines comprising the "Interpreter" in HERO's ROM monitor (roughly 7K bytes of HERO's 8K ROM constitute this Interpreter - the other 1K are devoted to other useful monitor subroutines, de-bugging tools and utility programs of one kind or another referred to above and detailed further on in this and later chapters). 6800 machine language (M.L.) Op Codes (6800 instruction set), on the other hand, are executed by the control section and the microprogram within the 6800 microprocessor - just as with every microprocessor.

Hence, when you write a program that contains both HERO R.L. interpreter command codes and 6800 M.L. Op Codes (as will almost always be the case) there must be some way that the R.L. codes are inhibited from going from their RAM locations (where you write your program) down the data bus to the control section/ microprogram of the CPU for "execution", but, instead, "find their way" to the right interpreter subroutine in ROM monitor where 6800 M.L. codes execute the robot command (R.L.) code to give the desired robot action. The true 6800 Op Codes in your program, however, should be allowed to go from their RAM locations in your program down the data bus to the control section/ microprogram for execution as regular 6800 M.L. Op Codes. How can that be accomplished? That is the detailed subject of Chapter 4.

Meanwhile, suffice it to say that the general rule is as follows: precede a stretch of R.L. interpreter commands in your program by the 6800 Op Code "3F" (SWI: Software Interrupt); precede a stretch of M.L. 6800 Op Codes in your program by the R.L. command 83 ("change to Machine Language") if coming out of a stretch of R.L. code. Note that 3F (SWI) is not a robot language command, yet it has the effect of "changing to Robot Language". Hence 3F is placed in parenthesis in the list of interpreter commands (Appendix A here, or p.115 Heath Technical Manual) as it is not an interpreter command but a legitimate 6800 Op Code. It is not one of the "37 Interpreter Commands" mentioned before. The whys and wherefores of the above remarks are thoroughly explored and explained in Chapter 4. That chapter is most important if you wish to know and understand the basis of any higher robot language and not merely content yourself with pushing keys to achieve robot actions (a robot operator). For the present, follow the rules to be explained in Chaper 4: use 3F before a stretch of R.L. robot language interpreter commands if coming out of a stretch of machine language (M.L.) Op Codes; and use 83 before a stretch of M.L. machine language Op Codes if coming out of a stretch of robot language (R.L.).

ENTERING A PROGRAM ON HERO

Strike the Reset key, then key 1 (or key A), then key A. See ----Ad on the display. Enter starting address of your program e.g. 0500. See 0500--. Enter first byte of your program e.g. 86. See 0501--. Enter second byte of your program e.g. FF and see 0502--on the display. Continue entering bytes e.g. BD,F7, 77,20,FE.

When this has been done you should see 0507--. You have entered the following (machine language) program:

```
0500      86 FF        LDAA FF (FF to ACCA)
  02      BD F7 AD     JSR F7AD (OUTBYT displays ACCA)
  05      20 FE        BRA FE (Halt)
```

RUNNING A PROGRAM

The above program starts in machine language. Hence it must be run as follows: press the Reset key, then key 1, then key D. See ----Do on the display. Then enter the starting address 0500. See the result demanded by this program i.e. FF on the left-most two digits of the display (OUTBYT monitor routine at F7AD displays the contents of the accumulator A, in this case FF. See the section on HERO's monitor subroutines later in this chapter).

If, however, our program started in robot language (e.g. C3: interpreter coded command for "motor move, wait, absolute (immediate)") such as the one below

```
0600      C3 50 20     Move arm to position 20 (HEX units).
  03      83           Change to machine language.
  04      20 FE        BRA FE (Halt). (Machine code).
```

you'd have to execute it by hitting Reset, key A, then key D. See ----Do. Then enter the starting address (0600 here). The program will execute and you'll see the arm (shoulder) move up to position 20 (HEX units) from its initial fully vertical position 00 (C3 50 20 does that, where the operand 50 invokes the shoulder stepper motor at medium speed and the operand 20 is the destination (absolute) position - more on the motor move commands like C3 later on - they are obviously most important to us in controlling HERO. If you tried to run the above program by entering keys 1,D,0600 after Reset, it would not execute (see Appendix B for details on the "motor move" commands).

EXAMINING AND MODIFYING A PROGRAM

To examine the first program at 0500 above, you'd press the Reset key. Then key 1 (or A), then key E (for "examine"). You'd see ----Ad. You enter the address of interest (here 0500). See 0500 86. Strike key F (the "forward" key) and see 0501 FF. The program is being displayed. To change FF at 0501 to AA, simply press the C key ("change"). See 0501--. Enter the replacement byte (AA). See 0501 AA. Press the F key and now see 0502 BD. AA has replaced FF at 0501. Examine the rest of the program by stepping forward with the F key, stopping only to change a byte in memory by pressing the C key, then entering the new byte and striking the F key. If you run this modified program (Reset, key 1, key D, 0500) you'll see AA on the display due to the call (JSR) of the OUTBYT subroutine in the monitor at address F7AD.

SINGLE STEP, BREAKPOINTS

Consult your Heath Robot Technical Manual pp.9 through 19 for details and examples on single stepping and the use of breakpoints as de-bugging tools for your programs, based on HERO's monitor.

A NOTE ON HERO's MEMORY ALLOCATION

RAM space: 0000 to 0FFF (4K).
ROM space: E000 to FFFF (8K).
User RAM: 003F to 0EE0.
RAM locations 0000 to 0006: each of these locations stores the seven stepper positions reached at any time: 0000 holds the arm's extend/retract position; 0001 the arm (shoulder's) swing position; 0002 the wrist rotate position; 0003 the wrist pivot position; 0004 the gripper opening; 0005 the head position; 0006 the wheel's steering orientation. In addition RAM location 0010 stores the number of ultrasonic target hits and 0011 stores the distance (HEX units) to the nearest target detected by the sonar on-board HERO. Each of these locations can be read (polled) at any time to monitor instantaneous individual stepper motor positions. We shall find the information stored at those locations extremely useful as we poll them in many of the experiments and applications in this manual.

THEORY OF OPERATION OF HERO

To gain the fullest appreciation of HERO's capabilities and of how and why it functions as it does (from a hardware standpoint), you are urged to carefully read pp.71-82 of your Heath Robot Technical Manual ("Theory of Operation"). In that connection, a study of HERO's schematics (part number 595-2872, parts 1 of 4, 2 of 4, 3 of 4, and 4 of 4) and of its "Block/Interconnect Diagram" (part of 595-2872-ET-18) will prove most rewarding. This suggested reading and study is not at all essential to the understanding and performing of the experiments in this manual, but "a little knowledge always helps."

APPENDICES A AND B IN THIS MANUAL

We shall frequently refer to the useful material in Appendices A and B at the end of the manual. A contains HERO's "Programmer's Information Sheet", the 6800 instruction set, and schematics of HERO's input and output I/O's with their addresses. Appendix B contains a very useful table (and how it is arrived at) of second byte (SS) operands that accompany motor move commands (first byte) for each of HERO's stepper motors and for each of the speeds desired (slow, medium, fast), as well as the third byte (XX) operands giving range of distances or positions allowable for each motor on HERO. The table will save much time and headache in your robot language programming endeavors.

EXPERIMENTS WITH SOME OF HERO's MONITOR SUBROUTINES

As mentioned earlier there are very useful and powerful subroutines in all monitors and HERO's is no exception. By calling them, we can add much to the real world power of our applications and experiments without having to "re-invent the wheel" each time. Those subroutines are to be considered as tools in our "tool box". We shall employ them frequently throughout this manual as they help both in easing our programming task and in making the applications and experiments more useful and realistic as they relate to real world situations. The most frequently employed are listed below, together with their location addresses in HERO's ROM. They are described as to what they accomplish when called and illustrated with a hands-on example. This chapter employs machine language exclusively.

A). INPUT A CHARACTER ("INCH" AT F777)

This subroutine, when called (by a JSR: Jump to Subroutine: BD Op Code), waits for user to enter a key. It then returns to user's program with the key's HEX code 0X in the accumulator (where X=0,1,....,F for keys 0,1,....,F respectively). An example of its use will be given in conjunction with the use of the subroutine OUTHEX discussed below in (B).

B). OUTPUT A HEX NUMBER (0,1,...F) TO THE DISPLAY ("OUTHEX" AT F7B5). EXPT. 1-1

OUTHEX, when called, sends the lower nibble of the accumulator to the display. An experiment demonstrating the use of both INCH and OUTHEX is given in the program below. This program, when run, waits for you to enter a key. As soon as you do, OUTHEX displays its code 0,1,....,F (or more accurately the lower nibble of its code 00,01,....,0F) on one digit of the display. Entering another key displays the lower nibble of its code on the next digit over to the right of the display, and similarly for a third entry, etc.

```
0200      BD F7 77      JSR INCH.  Waits till key entered.
  03      BD F7 B5      Display lower nibble of key code (JSR OUTHEX).
  06      01            NOP (no operation)
  07      01            NOP
  08      01            NOP
  09      20 F5         BRA F5 (branch to 0200).
```

We have inserted NOPs to make room for instructions to be added in Expt. 1-2 in section (C) below.

Running the program and entering 6 keys in a row will show the low nibble X of the key's code 0X on the six display LEDs successively from left to right. Entry of any new keys will not show up on the display. This is remedied by using the subroutine REDIS (Reset The Display) discussed in (C) below where each entry will show up on the left-most digit only and, therefore, any number of keys can be entered.

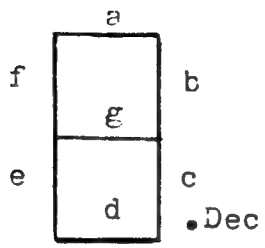
C). RESET THE DISPLAY ("REDIS" AT F64E). EXPT. 1-2

This routine resets the display address to the left-most LED (useful when the LEDs are filling up left to right as in Expt. 1-1 in (B) above).

If you now replace the NOPs 01 at 0206,7,8 in (B) above with BD F6 4E (JSR REDIS) and run the resulting program, you'll find that pressing any number of keys after you run the program will always show the low nibble X of the code 0X of the last key entered on the first display LED at the left. REDIS does not clear the display after the first nibble is displayed, but rather sets the display "pointer" to the left-most LED's address so as to get ready to display the nibble of the next key to be entered at the same LED.

D). OUTPUT A CHARACTER ("OUTCH" AT F7C8). EXPT. 1-3

Calling OUTCH allows the user to produce his own variety of characters, psuedo-characters, and shapes by loading the accumulator appropriately before calling OUTCH. The appropriate accumulator byte is determined as follows:



Byte = Dec a b c d e f g
MSB LSB

A 1 at any bit location in the above byte lights the corresponding LED's segment, a 0 blanks it. Thus, should you wish to realize the character L, you must choose d,e,f=1,1,1 and the other bits = 0. The accumulator must thus be loaded with the byte 0E=00001110 prior to calling OUTCH. The following program will accomplish this.

```
0200      86 0E          LDAA 0E. Load ACCA with byte 0E.
  02      BD F7 C8       JSR OUTCH. See letter L.
  05      20 FE          BRA FE. (Halt).
```

Change the byte at 0201 to output the character d (ɖ), - (hyphen), and a version of lower case r (ɾ). Thus, since you have control over each segment individually through the byte loaded into ACCA, you can produce, within limits, what you wish on the display.

The following program will display the word HEro. Note the bytes we use to load the accumulator for each letter. Verify them.

```
0200      86 37          LDAA 37
  02      BD F7 C8       JSR OUTCH
  05      86 4F          LDAA 4F
  07      BD F7 C8       JSR OUTCH
  0A      86 05          LDAA 05
  0C      BD F7 C8       JSR OUTCH
  0F      86 1D          LDAA 1D
  11      BD F7 C8       JSR OUTCH
  14      20 FE          BRA FE. (Halt)
```

A better way to output a string of characters is discussed in (E) below.

E). OUTPUT A STRING OF CHARACTERS. ("OUTSTR" AT F7E5). EXPT. 1-4

This routine displays the characters represented by the bytes listed in the program just after the instruction that calls OUTSTR. The last character in the string is recognized by the subroutine as such only if it has 80 added to the byte

that it would normally be (i.e. the decimal point bit in the byte for the LED is set to 1).

```
0200      BD F7 E5      JSR OUTSTR
  03      37            Byte for H
  04      4F            Byte for E
  05      05            Byte for r
  06      9D            Byte for o (with 80 added)
  07      20 FE         BRA FE. (Halt).
```

When you execute, you'll see HEro (with the decimal point to the right).

F). CLEAR DISPLAY (BLANKS IT). ("CLRDIS" AT F65B). EXPT. 1-5

This routine clears the display (blanks all segments) and sets the address in the display pointer to the left-most digit for new data to be displayed. Here's an example of its use:

In the last program (Expt. 1-4 in (E) above), replace 20 FE at 0207 with the following:

```
0207      BD F7 77      JSR INCH. Waits for any key to be pressed.
  0A      BD F6 5B      JSR CLRDIS. Blanks display.
  0D      BD F7 77      JSR INCH. Waits for any key to be pressed.
  10      7E 02 00      JMP 0200. Jump back to 0200.
```

When you execute this new program you'll again see HEro on the display. Then striking any key (except Reset) will clear the display. Pressing any key a second time will re-display the word HEro. You can continue this sequence of events by pressing keys one after the other and seeing HEro, blank, HEro, blank, etc.

G). OUTPUT A BYTE TO THE DISPLAY ("OUTBYT" AT F7AD). EXPT. 1-6

This routine is very useful, especially when we wish to display certain important parameters of an unfolding robot application e.g. the status of things as they are happening. We shall make a lot of use of it throughout this manual.

OUTBYT displays the full byte contained in ACCA (OUTHEX displayed only its lower nibble). An example of its use follows:

```
0200      86 EF         LDAA EF (EF to ACCA).
  02      BD F7 AD      JSR OUTBYT. Will display EF.
  05      20 FE         BRA FE. (Halt).
```

When this program is executed the display will show EF at its left-most two digits.

H). CALL OF "INCH" TWICE AND "OUTBYT" CREATES A FULL BYTE FROM TWO KEYBOARD ENTRIES AND DISPLAYS IT. (SEE ALSO PART (I) BELOW). EXPT. 1-7

Here we create our own useful routine which will allow entering a byte XY from the keyboard into ACCA by simply pressing the two keys X and Y (where X or Y 0,1, ...,F). Our program below then shows the byte on the display.

```
0200    BD F7 77    JSR INCH. Enter key X. Return from INCH with 0X in ACCA.
  03    48          ASLA. Arithmetic Shift Left on ACCA (each bit in A is
                    shifted one left, MSB of A goes into Carry).
  04    48          ASLA
  05    48          ASLA
  06    48          ASLA. X0 now in ACCA.
  07    16          TAB. Transfer (ACCA) to ACCB.
  08    BD F7 77    JSR INCH. Enter key Y. Return with 0Y in ACCA.
  0B    1B          ABA. Add ACCB (X0) to ACCA (0Y). XY now in ACCA.
  0C    BD F7 AD    JSR OUTBYT. XY is displayed.
  0F    01          NOP
  10    01          NOP
  11    01          NOP
  12    20 EC       BRA EC (branch back to 0200).
```

Execute the program. Enter 2 keys e.g. 5,6. See 56 on left two digits of the display. Then enter keys 8,9. See 89 to the right of 56 on the display. Enter A,B. See AB on the right-most two digits. Further entries will not show up on the display.

If you replace the NOPs at 020F,10,11 with BD F6 4E (JSR REDIS), then every time you enter 2 keys you'll see the corresponding byte on the left-most two digits of the display since REDIS resets the display's address pointer to the left-most LED each time it is called.

The above program is instructive, but fortunately it is already available as a subroutine in the monitor. The next section explains and uses that subroutine ("IHB" - Input Hex Byte).

I). INPUT AND DISPLAY HEX BYTE FROM THE KEYBOARD ("IHB"-INPUT HEX BYTE - AT F796). EXPT. 1-8

IHB, when called, does very simply what was done in section (H) Expt. 1-7 above - namely it monitors the keyboard, takes two keyboard entries (keys X,Y=0,1,....,F), forms the byte XY, stores it in ACCA and also displays that byte XY. Obviously, the program behind that subroutine must be identical to ours in Expt. 1-7 above. We'd use it as follows:

```
0200    BD F6 4E    JSR REDIS
  03    BD F7 96    JSR IHB. Enter any two keys X,Y and return with byte XY
                    in ACCA and displayed.
  06    20 F8       BRA F8. Branch back to 0200 to wait for entry of another
                    2 keys.
```

Run and enter any two keys X,Y. See the byte on the display. Unlike our routine in Expt. 1-7 (H above), IHB allows each nibble as it is entered to be displayed before the next one is entered.

J). DISPLAY BYTES ("DISPLAY" AT F6F9). EXPT. 1-9

Where the routine OUTBYT showed one byte on the display, DISPLAY here can show up to 3 bytes on the display. The bytes to be displayed come from consecutive locations in memory, the starting address of which is determined by the contents you previously place into the index register (X) - used here as an address pointer. The number of bytes to be displayed, starting at that address, must be loaded into ACCB before DISPLAY is called. Then calling address F6F9 will display the bytes in question (no more than 3 at a time).

As a useful example of DISPLAY, let us display information on the positions of the gripper's stepper, head stepper, and steering stepper on HERO (say in its initialized state-after you key in "31" for the utility program which does initialization of all motors after Reset). Earlier in this chapter under the section "A NOTE ON HERO's MEMORY ALLOCATION" we discussed the significance of RAM locations 0000 through 0006: they stored stepper positions of, respectively, arm extend/retract, arm (shoulder) swing, wrist rotate, wrist pivot, gripper opening, head position, and wheel steering orientation. Thus if we wish to display information on gripper, head, and steering positions, we could do so with the following program.

```
0200    BD F6 4E    JSR REDIS.  Left display.
  03    CE 00 04    LDX 0004.  Load index register with RAM address giving
                    gripper position.
  06    C6 03       LDAB 03.  Load ACCB with number of bytes (positions)
                    to be read.
  08    BD F6 F9    JSR DISPLAY.  This will show contents of addresses 0004,
                    0005, and 0006 on the display.
  0B    20 FE       BRA FE. (Halt).
```

When you run this program you'll see 00 62 49 on the display indicating HERO's initialized positions are 00 for gripper (closed), 62 for head (straight ahead) and 49 for steer (straight ahead). If you now change CE 00 04 to CE 00 00 and re-run, you'll see 00 00 4D on the display indicating arm extension is 00 (fully retracted), shoulder is at 00 (arm straight down), and wrist rotate is at position 4D (gripper is level). All the above assumes you've initialized HERO into its "nest" by invoking the utility program to do so after Reset (keys 3,1 will do that when pressed). Finally, if you change CE 00 04 to CE 00 03 and C6 03 to C6 01 and then execute, you'll see 00 on the display indicating the wrist pivot position to be at 00 (gripper horizontal).

The above will be very useful later on in keeping track "on the fly" of HERO's 7 different positions of its 7 different stepper motors (extend, shoulder, wrist rotate, wrist pivot, gripper, head, and steer).

This concludes Chaper 1. All the material studied in this chapter will prove extremely useful in the robot work we are about to embark on.

IMPORTANT NOTE

Looking down on HERO with the keyboard in front of you, if the shoulder swings up from right to left HERO is "right-armed". If it swings up from left to right it is "left-armed". Ours was "right-armed". If yours if "left-armed", a few programs later may have to be modified. We'll point out where.

CHAPTER 2

MACHINE LANGUAGE CONTROL OF HERO's I/Os

The information in (A) below presents HERO's I/O (memory mapped) addresses-both output and input ports. A brief description of each addressable I/O is appended. These descriptions will become more detailed as we work with the various I/Os in the experiments of this chapter.

Only by working with each I/O individually, and in machine language (at least to start), will we gain mastery, control, and confidence as to how they function and how they can be controlled. It will lay the basis for an appreciation of HERO's Robot Language (R.L.) that will make the task of controlling HERO's I/Os simpler than the machine language approach. HERO's R.L., through its "interpreter" program in ROM, makes life easier for us by replacing a number of (or many) machine language instructions to control I/Os by simple and succinct "interpreter command" codes defined (and implemented in machine language by the interpreter) to achieve the same control. Even so, machine level control of HERO's I/Os is often necessary as a companion, or alternative (or only choice) to R.L. to achieve maximum capability, versatility, flexibility and power. "Weaving" in and out from one language to another will prove most useful, and many times necessary, in our robot applications experiments. In many cases, what we do in this chapter by machine languuage (for teaching purposes and understanding), HERO's monitor (executive) and its interpreter will do for us more simply. And, indeed, in the chapters that follow, wherever we can use HERO's simpler R.L. instead of the more lengthy M.L., we shall certainly do so.

A). HERO's I/O ADDRESS AND THEIR DESCRIPTIONS

All addresses of I/Os on HERO are memory mapped. Thus any Op Code operation description in the 6800 instruction set that involves the letter M (M for memory) is applicable to a real memory address or to a physical I/O address-they are indistinguishable to the CPU. In what follows, consult Appendix A.

HERO's OUTPUT PORTS

The following lists HERO's output I/O addresses and gives a short description of each.

C2E0(OUT). To light/sound (bit 7), main power, sense, display, speech,

motion, sonar, tape out (bit 0) i.e. 8 output lines at OUTPORT C2E0. These 8 port bits are power enable/disable bits to the various I/O boards named above or to components thereon.

C2C0(OUT). The 8 output lines at this port are ARM SEL A (bit 7), ARM SEL B, speech strobe, spare (reserved) line, time clock (4 bits). The 4 clock bits tell the clock what we want to read. ARM SEL A line (when 1) selects either wrist rotate, shoulder, or head. ARM SEL B (when 1) selects either wrist pivot, gripper, or arm extend/retract. The speech strobe line strobes the phoneme speech byte placed on the data bus into HERO's VOTRAX phoneme synthesizer (it will do so when it is made to go low to high).

C2A0(OUT). Main drive control: forward/reverse; on/off; 6 speed bits for main drive motor (body motion).

C280(OUT). 4 upper bits drive either the arm extend/retract stepper or the head rotate stepper; the other 4 lower bits drive either the shoulder (arm lift) stepper or the gripper stepper.

C260(OUT). 4 upper bits drive steering stepper for main drive motion; the other 4 lower bits drive either the wrist pivot or wrist rotate stepper.

C240(OUT). 8 bits go from C240 to the speech board. The two high bits determine pitch, the other 6 bits comprise the phoneme code (64 such codes) constituting an element of speech. The byte output to C240 inputs to the VOTRAX SC01 phoneme speech synthesizer board to create the proper phoneme sound.

C220(OUT). 8 latched data lines available as an OUT port on HERO's experimental board where you can "hang" LEDs or any other output I/O external to HERO.

C200(OUT). 8 output lines that can clear the interrupt latches so that further interrupts can be made and honored. The 8 interrupts involved are INT on the experimental board (bit 7), wheel detect, trigger, time clock, battery low (logic board), battery low (motor), motion, and sonar (bit 0).

C300(OUT). Time clock. These bits output to C300 tell the clock what we wish to do i.e. read or write.

C1YX(OUT). Address of a segment on one of the 6 displays above HERO's keyboard. Y=1, 2,3,4,5,6;X=0,1,2,3,4,5,6,7 i.e. 8 segments including the decimal point. (See the figure in Expt. 3-7).

HERO'S INPUT PORTS

C2A0(IN). Experimental board inputs (tri-stated). You can add input I/O's external to HERO such as switch bits, an A to D converter, etc.

C280(IN). Sleep/Normal switch and remote control (teaching pendant) inputs (speed,

left/right, motor direction, arm or base, trigger controls on the pendant). These can all be polled.

C260(IN). Limit switch inputs, tape-in, and speech request (from voice synthesizer board). Limit switches can tell us (when input to CPU) when steering reaches CCW or CW limits, or when head reaches CCW or CW limits, or when arm is full-up or full down.

C240(IN). Sense inputs. Port C240 is the A to D converter inputting the byte representing analog level of light intensity or sound level depending on whether the light or sound transducer circuit has been enabled.

C220(IN). Sonar timer inputs. Port C220 receives the ultrasonic ranging system's timer-counter bytes at any instant. This timer-counter is used by the ultrasonic ranger to measure the passage of time between transmission and reception of an ultrasonic pulse. This time interval is a measure of distance to the target causing the echo. The sonar timer-counter bytes are available to the user if he inputs the byte from C220 to CPU. (In the ultrasonic ranging mode of operation the CPU is made to read C220 for us when a range reading is to be taken. The value at port C220 is then a measure of range).

C200(IN). Interrupt input port. This port holds 8 input bits each of which represents the source of a different interrupt (a request from INT on the experimental board (bit 7), or from the wheel (pulse disc input from its optical pick-up), or from trigger on the teach pendant, or from the time clock, or from a low voltage condition on logic or motor boards, or from the ultrasonic detector motion board indicating motion, or from the sonar (bit 0)). This port would have to be polled as to the source of the interrupt since all are subsequently NANDed together to form one interrupt line into the CPU's IRQ interrupt pin which, aside from NMI, is its only input interrupt pin.

C300(IN). Clock input data. Gives time when read. Can be displayed.

B). EXPERIMENTS USING HERO's I/O PORTS

The purpose of this section is to gain total familiarity with HERO's I/O ports by controlling them through write (output) instructions (B7 XY UV where B7 is the Op Code for STAA (Store ACCA) and XYUV is the output I/O's (or memory's) address at which ACCA is to be stored (written)); or by polling them through read (input) instructions (B6 X'Y' U'V' where B6 is the Op Code for LDAA (Load ACCA) and X'Y'U'V' is the input I/O's (or memory's) address which is to be read (loaded) into ACCA). In other words, we shall work at the machine language level where we'll know exactly what we're doing, and why, at every stage of controlling or communicating with an I/O. Later on, of course, we'll be employing HERO's interpreter which will allow short and succinct Command Codes ("Robot Language" codes) with operands (if any are needed) to be employed to achieve certain I/O control. That, of course, will, in a good number of instances, achieve I/O control in much easier fashion. Still, there will be many instances where you'll have to fall back on the machine

language approach we are presenting in this chapter to get the detailed I/O control that the robot language may not always give us.

The experiments that follow will work with the I/Os in a different order from which their addresses were listed in section (A) above.

EXPERIMENT 2-1

WORKING WITH THE EXPERIMENTAL BOARD (OUTPORT C220, INPORT C2A0)

We have added four LEDs to experimental board outport C220 to receive bits D3,D2,D1, D0 output from the CPU, and four switches to experimental board inport C2A0 to input bits D3,D2,D1,D0 to the CPU (you can, of course add 4 additional LEDs and/or switches at those ports to involve the full data byte D7....D0). Port C220 is already latched and port C2A0 is already tri-stated (see figures in Appendix A). The addition of LEDs to port C220 and switches to port C2A0 can be accomplished as follows: between ground on the experimental board and the cathode side of each of four LEDs, connect four 220 ohm resistors. Each of the LED's anode sides should be connected to DO3,DO2,DO1,DO0 on the experimental board. As for the switches, one side of each switch (DIP package) should be connected to +5V on the experimental board and the other sides to DI3,DI2,DI1,DI0 on the experimental board and to 1K resistors between DI3,DI2,DI1,DI0 and ground.

Here are two examples on outputting and inputting to and from these ports:

```
0200     .BD F7 77      JSR INCH. Enter a key X.
  03     B7 C2 20       Output code 0X to C220.
  06     20 F8          BRA F8 to 0200.
```

When the above program is executed the LEDs will show the low nibble X of any key that you press at any time. Thus if 0,5,7,A, and F are pressed, you'll see, in turn, 0000,0101,0111,1010, and 1111 at the LEDs.

If you replace BD F7 77 by B6 C2 A0 where B6 C2 A0 means LDAA C2A0 i.e. read the switches at port C2A0, and re-run, then you'll see the status of the switches S3,S2, S1,S0 at the LEDs as you throw them hi/lo/hi,....etc. Thus S3,S2,S1,S0=1,0,0,1 will show up as 1,0,0,1 on the LEDs; 1,1,0,0 will show as 1,1,0,0, etc.

WORKING WITH OUTPORT C2E0

Outport C2E0 latches 8 bits output from ACCA in the CPU to the data bus if the instruction B7 C2 E0 is employed in your program (B7 is the Op Code for STAA: Store ACCA at the address specified). Remember, all input and output I/Os "hang" on the same data bus. Decoders are therefore needed in the I/O interfacing (implemented for us in HERO) so that the address placed on the address bus by the CPU in the execution of the relevant I/O instruction, together with the action of the decoder and read or write signals emanating from the CPU, will latch data to the rele-

vant (addressed) output I/O in question (or read data from it if it is an input I/O).

The 8 bits latched at C2E0 are then routed to various enables on various I/Os of HERO, turning power on or off at those I/Os i.e. activating or de-activating them according as they are 1 or 0. The various latched bits at C2E0 are routed as follows (see I/O block diagrams in Appendix A):

D7: 1 enables light detector circuit, 0 enables sound detector circuit (one or the other).
D6: Main power enable. 1 disables power, 0 enables.
D5: Sense board enable (ADC for light or sound digital input). 1 enables, 0 disables.
D4: Display enable. 1 enables, 0 disables.
D3: Speech board (speech synthesizer) enable. 1 enables, 0 disables.
D2: Motion detect board enable. 1 enables, 0 disables.
D1: Sonar board enable. 1 enables, 0 disables.
D0: Tape out enable. 1 enables, 0 disables. (Cassette tape).

Thus C2E0 is a key output I/O, as the 8 bits output to it are used as power on/off (enable/disable) bits for the crucial I/Os listed above. We shall now embark on a series of experiments demonstrating the use of I/O port C2E0 in enabling or disabling the I/Os controlled by D7....D0 above. These experiments will also give us a chance to work with those I/Os when they are enabled - hence our first encounters with HERO's internal I/Os.

EXPERIMENT 2-2

USING OUTPORT C2E0 TO ENABLE THE SOUND SENSOR. READING SOUND LEVEL FROM SENSE INPORT C240.

In this experiment we shall enable the sound sensor and the sense board and then read (input) the digital (byte) level of sound from sense board input port C240 (see block diagrams in Appendix A). Port C240 is the analog to digital converter that converts analog light or sound level to its equivalent digital byte. We shall then show the HEX value of that level on the display. Hence we'll have to output a 0 to D7 and a 1 to D5 of port C2E0 as the discussion in the previous section shows. Moreover, since we intend to display sound level, we'll have to output a 1 to D4 of port C2E0. Therefore the byte we must output to C2E0 should be (at least) 30. The program follows:

```
0300    86 30       LDAA 30.  30 to ACCA.
  02    B7 C2 E0    STAA C2E0.  Enables sound, sense (ADC) board, and display.
  05    B6 C2 40    LDAA C240.  Input ADC byte from sense port.
  08    BD F6 4E    JSR REDIS.
  0B    BD F7 AD    JSR OUTBYT.  Display sound level.
  0E    CE 28 00    LDX 2800 (X=index register).
  11    09          DEX.  Decrement X (for time delay).
  12    26 FD       BNE FD.  Loop to 0311.
  14    20 EF       BRA EF to 0305 and poll sound again.
```

The instructions at 0300-04 enable sound detector, ADC sense board, and HERO's display. The instructions at 0305 through 030D read and display the sound level. The instructions at 030E through 0313 constitute a short time delay (about 1/6 sec) before another polling at 0305 is made (BRA EF). This time delay is achieved by loading register X and decrementing it till it's zero.

When you execute this program and speak loud/soft you'll see bytes vary on the display from high to low. When you stop talking you'll see 00 on the display in a quiet environment.

If now you change 30 at 0301 to 10 or to 20 you'll not see sound level displayed. Why? Or if, with 30 back at 0301, you run from 0305 you'll also not see display of sound level, why? Do you notice any difference on the display if you run from 0300 with first 20, then 10, at 0301?

With 30 back at 0301, run the program from the following "patch":

```
0320      3F          SWI INTR.  Change to Robot Language.
  21      42          Robot Language command code for "enable the sound detector"
  22      83          R.L. command code for "change to Machine Language".
  23      7E 03 05    JMP 0305.  Jump to 0305.
```

When you execute the program from 0320, all will work even though 7E 03 05 jumps to 0305 and bypasses the 86 30, B7 C2 E0 sound and sense board enable instructions. That's because the R.L. (robot language) code 42 at 0321 ("enable sound detector") does the job of enabling the sound detector and sense board. Hence we conclude that 86 30, B7 C2 E0 machine language code for enabling sound detector and sense board are equivalent to the simple R.L. interpreter command 42.

EXPERIMENT 2-3

USING PORTS C2E0 AND C240 TO ENABLE LIGHT DETECTOR AND READ LIGHT LEVEL.

In this case we must output a 1 to D7, a 1 to D5, and a 1 to D4 of port C2E0. Hence, the only change needed in the last program is to replace 30 at 0301 by B0. Make this change and execute the new program. Use a light source at HERO's light sensor (LDR) input and see varying light level bytes on the display depending on proximity to the light source. Now replace B0 at 0301 first by 90 then by A0 and execute in each case. You'll not see light level bytes on the display. Why? With B0 back at 0301 and 41 (Robot Language command code for "enable light detector") replacing 42 at 0321, again run from 0320 thereby by-passing the 86 B0, B7 C2 E0 light and sense board enabling instructions. All will work. Conclusion: R.L. interpreter command 41 does what 86 B0,B7 C2 E0 did - namely, enable light detector and sense board.

EXPERIMENT 2-4

USING OUTPORT C2E0 TO ENABLE DISPLAY

We have enabled and disabled the display in the previous two experiments (2-2 and 2-3) by outputting either a 1 or a 0 to D4 at port C2E0. To see the connection between R.L. interpreter commands 4E ("enable display") and 5E ("disable display") and the machine language instructions that do the same, proceed as follows:

```
0300    86 10       LDAA 10.  D4=1.
  02    B7 C2 E0    STAA C2E0.  Enable display.
  05    86 FF       LDAA FF
  07    BD F7 AD    JSR OUTBYT.  See FF.
  0A    20 FE       BRA FE. (Halt).
```

When the above program is executed, you'll see FF on the display. If, however, you replace 10 at 0301 with 00 and run, you'll not see FF, as D4=0 at C2E0 disables the display. To achieve the same results with the R.L. interpreter commands 4E and 5E, proceed as follows (executing from 0310):

```
0310    4E          Enable display.
  11    83          Change to M.L.
  12    86 FF       LDAA FF
  14    BD F7 AD    JSR OUTBYT
  17    20 FE       BRA FE. (Halt)
```

Execute the above by pressing Reset, key A, key D, and 0310. You'll see FF on the display. .If you replace 4E at 0310 by 5E ("disable display"). You'll not see FF. Conclusion: 86 10, B7 C2 E0 machine code is equivalent to interpreter command (R.L.) 4E as far as enabling the display. And 86 00, B7 C2 E0 machine code is equivalent to R.L. interpreter command 5E in disabling the display.

EXPERIMENT 2-5

USING OUTPORT C2E0 TO ENABLE HERO's MOTION DETECTOR.DETECTING MOTION.

Looking back at the meanings of D7...D0 at port C2E0, we see that a 1 must be output to line D2 in order to enable the motion detect board. Bit D2 ("motion") at C2E0 controls the enable line into the motion circuit board. This motion board works on the basis of ultrasonic transmission and reception such that any motion of a detected target causes AM modulation on the received wave. This AM modulation is detected and produces an INTERRUPT signal out of the motion circuit board to the bit 1 line of input port C200 (which is HERO's interrupt input port - see Section A this chapter on I/O addresses and descriptions). This interrupt port is polled by the monitor at address EFC9 in the monitor. (This comes about when the NANDed output of interrupt port C200, (see p.119 of your Technical Manual) feeding into the IRQ interrupt pin of the 6808, causes an IRQ interrupt request which, with the 6800

or 6808 CPU, causes a program counter vectoring to memory address FFF8, FFF9 where, in the case of HERO's ROM, the bytes EF and C9, respectively, are found and loaded into the PC). When the monitor polling that commences at EFC9 finds that the source of an interrupt was motion (i.e. the reading of port C200 into ACCA (B6 C2 00) would produce 02 in that case since motion detect interrupt is bit 1 of inport C200), it (the monitor) jumps to 0027 in user RAM where 3 bytes are allowed the user (at 0027,28, and 29) to jump himself to his motion interrupt service routine. (NOTE: if the source of an interrupt is the trigger on the remote control pendant (20 would then be read at inport C200) the monitor jumps to 002A in RAM where again the user has 3 bytes to jump himself to his interrupt service routine. If the source of interrupt were INT on the experimental board (80 would then be read at inport C200), user is jumped by the monitor to 002D in RAM where 3 bytes of a jump instruction must be entered. In other words, the monitor's polling routines branch on 02 or 20 or 80 etc. (that were read into ACCA) to appropriate addresses in each case. More on all these matters when we treat interrupts in detail in Expt. 3-4).

Hence, to service a motion interrupt, our program must itself write the bytes 7E UV WX out to addresses 0027,28,29 in RAM where 7E is the JMP Op Code and UVWX is the address where we wish to be jumped to for the interrupt service routine we shall write. In the program below we take UVWX to be 0520. Since our service routine will use the display and we wish to have the motion board enabled for this experiment, we need to output 1s to bits D2 and D4 of outport C2E0. This will be accomplsihed if we output byte 14 to C2E0. Our program follows:

```
0500      86 14          LDAA 14.
  02      B7 C2 E0       STAA C2E0. Enables display and motion board.
  05      86 7E          LDAA 7E.  7E to ACCA.
  07      B7 00 27       STAA 0027.  7E to 0027.
  0A      CE 05 20       LDX 0520.  0520 to X register.
  0D      DF 28          STX 28.  Store (X) at 0028 and 0029 i.e. 05 to 0028 and 20
                         to 0029.  DF (STX 28) is direct addressing mode.
  0F      0E             CLI.  Clear interrupt (IRQ) mask.  Enables IRQ line.
  10      20 FE          BRA FE.  Halt.  Self loop.
```

If motion is detected, the motion IRQ interrupt will occur during the self loop at 0510 and vector us to 0027,28,29 where 7E 05 20 is now found, causing a jump to 0520. At 0520 we shall write the following motion interrupt service routine:

```
0520      86 FF          LDAA FF.
  22      BD F7 AD       JSR OUTBYT.  See FF on display.
  25      39             RTS.  Return to loop at 0510.
```

If we execute from 0500 and initiate motion in the vicinity of HERO we'll see FF appear on the display at the moment motion is detected. In fact we'll see FFFFFF clear across the display because of multiple motion interrupts and the absence of REDIS (BD F6 4E) before JSR OUTBYT. If, now, you replace 0E at 050F by 0F(SEI: set the interrupt flag, thus disabling interrupts) and re-run, motion interrupts will not be honored and you'll not see FF on the display. Put back 0E at 050F and change 14 at 0501 to 10. This will cause 0 to be output to bit 2 of port C2E0 thereby disabling the motion board. If you execute this modified program, motion interrupts will be ignored. FF will not show on the display.

Consider, now, the following "patch" at 0530:

```
0530     3F          Change to robot language.
  31     4B          Interpreter command "Enable Motion Detector".
  32     83          Change to machine language.
  33     7E 05 05    JMP 0505.
```

With 14 back at 0501, execute the program from 0530 (Reset, A,D,0530). Motion will be detected and you'll see FF on the display even though you now bypass 86 14, B7 C2 E0 with the JMP to 0505 at 0533. We conclude, therefore, that the 4B robot language command "enable motion detector" is equivalent to our machine language instructions 86 14, B7 C2 E0, and 0E (CLI).

Next replace 14 at 0501 with 10 and 4B at 0531 with 5B (interpreter command code for "disable motion detector"). Running from either 0500 or 0530 and initiating motion will not produce FF on the display in either case. We conclude that 5B is the interpreter command equivalent to our machine code 86 10, B7 C2 E0 that disables the motion circuit board. Note that even if the motion board is enabled the display will not show FF if 14 at 0501 is changed to 04 and the program is run from 0500. Why?

EXPERIMENT 2-6

USING OUTPORT C2E0 TO ENABLE OR DISABLE MOTORS (MAIN POWER).

A review of the bit assignments at outport C2E0 in the section just after Expt. 2-1 ("Working With Outport C2E0") shows that bit D6 is the controlling bit as far as main power is concerned. We found that on our HERO D6=1 disables power to motors and D6=0 enables it. We shall use the motor interpreter command (R.L.) with the format D3 SS 20 where SS determines which motor is to be involved, 20 is the relative distance we wish to have it move, and D3 is the interpreter command (code) for "motor move, wait, relative (immediate)". See Appendices A and B.

The byte SS pertinent to the various motors and attendant speeds is found from SS= mmmssDdd where mmm=000 (drive motor), 001 (extend motor), 010 (shoulder motor), 011 (wrist rotate motor), 100 (wrist pivot motor), 101 (gripper motor), 110 (head motor), 111 (steer motor). ss are the speed bits: ss=01 (slow), 10 (medium), 11 (fast) (shoulder (arm) has only slow/medium speeds). The D bit determines direction of motor rotation in relative motion while dd can be taken 00 (it is used only to extend the drive (base motion) distance and does not affect the other motors). See Appendices A and B.

Thus for medium speed we have SS=10 (drive), 30 (extend), 50 (shoulder), 70 (wrist rotate), 90 (wrist pivot), B0 (gripper), D0 (head), F0 (steer). See table Appendix B.

Since we are going to drive various of HERO's motors, we shall use the convenient R.L. language interpreter commands (we could do it the "hard" way with straight machine language, and we do that just one time for learning purposes in Expt. 2-14 later in this chapter). Thus in the program below we will use the D3 motor move robot language command to do the job easily. The program follows.

```
0400    86 50        LDAA 50.  50 to ACCA.
  02    B7 C2 E0     STAA C2E0.  Outputs 1s to D6 (main power disabled) and D4
                     (display enabled).
  05    3F           SWI.  Change to robot language.
  06    D3 SS 20     Motor move, wait, relative (immed).
  09    3A           Robot language for "return to executive" ("ready").
```

Take SS, in turn, equal to 10,30,50,70,90,B0,D0, and F0 and execute in each case. You'll find that the pertinent motor does not move (at least on our HERO). 50 to C2E0 provided a 1 to D6 (main motor power) disabling the motor. Now change 50 at 0401 to 10. This causes D6 to be 0 when the instruction at 0402 is executed. That will cause main motor power to be enabled. Again take SS to be 10, 30, 50,70,90,B0,D0 and F0 and run each case. The appropriate motor now does move in every case. Thus the D6 bit of outport C2E0 must be 1 to disable main power and 0 to enable it.

EXPERIMENT 2-7

USING OUTPORT C2E0 TO ENABLE THE SPEECH BOARD. CREATION OF PHONEME SOUNDS USING OUTPORTS C240 AND C2C0.

Bit 3 (D3) at outport C2E0 must be made hi (1) in order to enable the speech circuit board (containing the VOTRAX phoneme synthesizer SC01 and fed with phoneme bytes (codes) from outport C240). Thus to create an audible phoneme sound we must first output a 1 to D3 of port C2E0 and then in addition input a phoneme code (byte) (say from the keyboard), then output same to port C240 which latches and inputs same to the VOTRAX phoneme speech board synthesizer. Not only that, but we must cause the strobe line into the speech board to make a 0 to 1 transition to latch the phoneme byte from port C240 into the VOTRAX synthesizer. This strobe line emanates from bit D5 of outport C2C0 which means we must output first a 0, then a 1 to bit D5 of port C2C0. (You are urged to consult our Appendix A, part A of this chapter on "HERO's I/O Addresses And Their Descriptions", your "Block/Interconnect Diagram" supplied with your technical manual, and your "Schematic of the Speech Accessory Circuit Board").

So here we have to control three output ports: C2E0 (to enable speech board); C240 (to output the phoneme code to that port); and C2C0 (to cause its output bit D5 to go lo to hi so as to strobe the phoneme code arriving from port C240 into the phoneme synthesizer board to create sound).

The following porgram does all of the above. The phoneme code will be input from the keyboard at run time by entering the two keys that determine the desired phoneme code.

```
0600    86 18        LDAA 18.  18 to ACCA.
  02    B7 C2 E0     STAA C2E0.  18 at C2E0 will enable display and speech board.
  05    BD F7 96     JSR IHB.  Waits for press of 2 keys X,Y.  Shows byte XY on
                     the display and stores same in ACCA (our phoneme).
```

```
  08      B7 C2 40      STAA C240. Outputs the phoneme XY to C240 for input to
                        synthesizer.
  0B      86 00         LDAA 00.
  0D      B7 C2 C0      STAA C2C0. Outputs 0 to D5 of port C2C0 (speech strobe).
  10      86 20         LDAA 20
  12      B7 C2 C0      STAA C2C0. Raises D5 at port C2C0 to a 1 to cause strob-
                        ing of phoneme byte at C240 into speech synthesizer.
  15      20 FE         BRA FE. Halt.
```

Execute the program and enter 2 keys e.g. 1,1. This creates the phoneme byte 11 which is the code needed to create the sound SH as in "SHop" (see p.80 of your Heath Voice Dictionary). Execute again and enter the keys 2,B. 2B will create the phoneme sound R as in "Red". Re-run and in each case try phoneme codes 10 (for CH), 18 (for L), 26 (for O), etc. In each case you hear the appropriate sound and see the phoneme byte on the display.

Now replace 18 at 0601 by 10. This will disable the speech board and no sounds will be heard when you execute and enter two keys. Put back 18 but replace 20 at 0611 by 00. Execute and try to produce sounds. It will not work because we now fail to bring the strobe line high.

You might now try sandwiching in 3F (change to R.L.), 05 ("abort speech" robot language command), and 83 (convert back to machine language) between 0604 and 0605 in the above program. If you run this modified program and again enter phoneme codes from the keyboard you'll hear no sounds. Why?

Another programming approach to this problem would be to have the phoneme bytes stored in memory and then accessed by indexed addressing for outputting to port C240 as before. An example of this approach follows (we store the phonemes 11,1F, 2B in a table starting at 0530).

```
0500      CE 05 30      LDX 0530. Base address of table loaded into X reg.
  03      86 18         These two lines of code enable speech board and display.
  05      B7 C2 E0
  08      A6 00         LDAA 00. Read byte from memory pointed to by X reg. plus
                        offset (00 here) into ACCA. Indexed addressing.
  0A      B7 C2 40      STAA C240. Output phoneme fetched above to C240 for
                        synthesis.
  0D      86 00         These four lines of code cause strobe (D5) line of C2C0 to
  0F      B7 C2 C0      go lo to hi to strobe phoneme at C240 into VOTRAX.
  12      86 20
  14      B7 C2 C0
  17      08            INX. Increment index register.
  18      BD F7 77      JSR INCH. Waits till any key is pressed and returns.
  1B      7E 05 08      JMP 0508.

0530      11            Phoneme for "SH"
  31      1F            Phoneme for "S"
  32      2B            Phoneme for "R"
```

EXPERIMENT 2-8

USING OUTPORT C2E0 TO ENABLE OR DISABLE THE SONAR BOARD (ULTRASONIC RANGING)

Bit D1 ("Sonar") at outport C2E0 controls the "enable" line into the sonar transmitter circuit board (see Appendix A, section A this chapter, and your "Block/Interconnect Diagram"). The latter board, together with the sonar receive circuit board, provides the electronics for ultrasonic range measurements of distance to targets (see also p.7-44 Fig. 7-29 of your Heath course notes "Robotics and Industrial Electronics"). The results of those measurements are constantly updated by the monitor and stored in RAM memory location 0011 (with number of target hits stored at 0010). To enable the ultrasonic ranging system requires, therefore, that a 1 be output to D1 of port C2E0. Since our program will want to show target range on the display, we'll also need to output a 1 to line D4 of port C2E0 to enable the display. The following program will show the relative range in HEX units to a target as it moves back and forth in front of the transmit/receive ports in HERO's head.

```
0400     86 12        LDAA 12.
  02     B7 C2 E0     STAA C2E0.  Enables sonar board and display.
  05     B6 00 11     LDAA 0011.  Read target range from RAM location 0011.
  08     BD F6 4E     JSR REDIS.
  0B     BD F7 AD     JSR OUTBYT.  Display range.
  0E     CE 20 00     LDX 2000 for short time delay (1/8 sec).
  11     09           DEX.
  12     26 FD        BNE FD.  T.D. not over.  Back to 0411.
  14     20 EA        BRA EA.  T.D. over.  Back to 0400 to poll target range.
```

Execute and hold a book or pad in front of the sonar transmitter/receiver. Move it back and forth. Notice the HEX "distance to target" reading on the display increase and decrease as the target recedes and approaches. Re-do the experiment with byte 10 replacing 12 at 0401. This will disable the sonar board and when you execute you'll see no variation in target range on the display as the target moves about other than a constant initial value read from 0011. With 12 back at 0401 run, move target, and then reset. Note the last range on the display at the time of resetting. It should agree with the value you read at 0011 when you directly examine the latter RAM memory location.

Let us now enable or disable or disable the sonar ranging system by using R.L. commands 45 ("enable ultrasonic ranging") or 55 ("disable ultrasonic ranging") in lieu of machine language enable/disable via bit D1 at port C2E0. To do this, add the following "patch" (after changing EA at 0415 to EF: why?):

```
0420     45           Enable sonar.
  21     83           Change to M.L.
  22     7E 04 05     JMP 0405 to poll range.
```

Run from 0420 (Reset, key A, key D, 0420). The ranging system works as target moves, even though JMP 0405 short circuits instructions at 0400 and 0402. We conclude that R.L. command 45, when executed by HERO's interpreter must be equivalent to M.L. enabling instructions 86 12, B7 C2 E0 used previously. The interpreter executes precisely that M.L. for us. Now replace 45 at 0420 by 55. Run from 0420 again.

The ranging system does not work. We conclude that 86 10, B7 C2 E0 (M.L. code) to disable the sonar is exactly what HERO's interpreter is doing for us when it executes the R.L. command 55. The interpreter translates it to the appropriate machine language.

EXPERIMENT 2-9

READING THE SONAR TIMER AT INPUT PORT C220.

It is suggested that you re-read the description of input port C220 (the sonar timer-counter) in section A of this chapter under "HERO's Input Ports". The program below reads and displays the last sonar time-count byte at the time any key is entered from the keyboard. Note that we need to enable the sonar transmit board by outputting a 1 to bit D1 at outport C2E0 (recall that D1 on C2E0 feeds the sonar board enable input (see "Working With Outport C2E0" earlier in this chapter and your "Block/Interconnect Diagram)). We shall be using the display, so the D4 line of C2E0 (display enable) must be set to 1. Hence we need to output the byte 12 to C2E0, just as was the case in the last experiment.

```
0800    86 12       LDAA 12.
  02    B7 C2 E0    STAA C2E0
  05    B6 C2 20    Read sonar timer byte (count).
  08    BD F6 4E    JSR REDIS.
  0B    BD F7 AD    JSR OUTBYT. Will display sonar timer byte count.
  0E    BD F7 77    JSR INCH. Waits till any key is pressed.
  11    20 F2       BRA F2. Poll timer again at 0805.
```

Execute and press any key at any time one after another. The display will show the last time-count byte at the time a key was pressed.

EXPERIMENT 2-10

USING INPORT C260 TO INPUT THE STATUS OF THE LIMIT SWITCHES

Let us first describe the pin-out of input port C260 (see Appendix A here and p.119 of your technical manual. There may be mistakes in your copy of the latter. We have corrected them according to the "Block/Interconnect Diagram" in what follows below). The inputs D7...D0 to C260 described below are held by C260 in the tristated condition until a B6 C2 60 read (LDAA) instruction de-tristates C260 and sends the bits D7...D0 into ACCA of the CPU via the data bus to which C260 is connected. The bits D7...D0 held by C260 are the following physical entities:

D7: Tape-in (cassette tape).
D6: Limit steer (CCW). 0 when limit reached, 1 otherwise.
D5: Limit arm (down). 0 when limit reached, 1 otherwise.
D4: Limit head (CCW). 0 when limit reached, 1 otherwise.

D3: Limit steer (CW). 0 when limit reached, 1 otherwise.
D2: Limit arm (up). 0 when limit reached, 1 otherwise.
D1: Limit head (CW). 0 when limit reached, 1 otherwise.
D0: Speech request (from voice synthesizer/speech circuit board).

The limit switches tell us when and if the corresponding position limits of the entity involved have been reached (by the fact that the limit switch inputs a 0 instead of the usual 1). The nominal ranges for steer, arm, and head with HERO are

Steer: 00 (CCW) to 93 (CW)
Arm: 00 (down) to 86 (up)
Head: 00 (CCW) to C2 (CW)

The upper limits may vary somewhat on your particular unit.

The robot language interpreter command code FD moves all motors to the absolute positions specified in the immediate 7 bytes (7 steppers - 7 degrees of freedom). Thus FD 98 86 93 A5 75 C2 93 would extend the arm to 98, swing the arm up to 86, rotate wrist to 93, pivot wrist to A5, open gripper to 75, rotate head to C2, and rotate steer to 93 (these bytes are the nominal maximum limits on the respective 7 steppers: arm extend, arm swing, wrist rotate, wrist pivot, gripper, head and steer in that order. We could, of course, employ FD with any immediate 7 bytes provided they don't exceed the limits just given: 98,86,93,A5,75,C2,93. In that case the motors would go to some intermediate position).

A program to read the limit switches and see if their settings (1 or 0) are consistent with the position limits given above is presented below.

```
0800    FD 05 07    Motors, move all abs (immed.) to 05,07,09,11,13,C2,03
        09 11 13    (extend...steer).
        C2 03
  08    83          Change to M.L.
  09    BD F7 77    JSR INCH. Wait for press of any key.
  0C    B6 C2 60    LDAA C260. Read port C260 into ACCA (limit switch bits).
  0F    BD F6 4E    JSR REDIS
  12    BD F7 AD    JSR OUTBYT. Show (C260) = limit switch status on display.
  15    BD F7 77    JSR INCH. Press any key to escape.
  18    3F          Change to R.L.
  19    3A          Return to monitor ("ready" spoken).
```

Execute the above program. See the robot achieve the various positions requested in the FD command and stop. Press any key. The display should, and will, read FD= 11111101 (a coincidence with the command FD), indicating D1=0 i.e. the head has reached its CW limit C2 and so its limit switch inputs a 0. Press any key and HERO says "ready". Now enter the utility motor initialization mode 32 by pressing keys 3,2. The utility resource in the monitor is invoked which returns HERO to its initial positions given by the bytes 00,00,4D,00,00,62,49 = arm extend, arm swing, wrist rotate, wrist pivot, gripper, head, steer positions. (Check these after initialization is complete by reading RAM memory locations 0000 through 0006. They will confirm these values).

Now replace C2 at 0806 by 00 (head CCW requested). Run and press any key after all robot motion ceases. The display will show EF=11101111 indicating D4=0 as to be expected since the head has reached its CCW limit 00. Press any key, then keys 3,2 to re-initialize all motor positions.

Next replace 07 at 0802 by 00 (arm vertically down) and change C2 (head rotate) to 62 which is HERO's straight ahead head orientation (half way between 00 and C2). Execute. Strike any key when all motion ceases and see DF=11011111 indicating D5=0 as expected since the arm is at its vertical (down) limit 00. Press any key and then keys 3,2 to re-initialize.

Replace 07 at 0802 by 86 (arm at its up limit) and C2 at 0806 by 62. Execute and press any key when robot motion stops. You should read FB=11111011 on the display, indicating D2=0 i.e. limit on arm up position (86) was reached. Again press any key, then keys 3,2 to re-initialize.

With 07 back at 0802 and 62 replacing C2 at 0806, change 03 at 0807 to 93 (CW steer limit i.e. maximum right turn). Execute. Press any key after motion stops. Display should show F7=11110111. D3=0 indicates limit on steer CW (93) was reached as requested. Re-initialize.

Now change the bytes at 0800-0807 to FD 98 86 93 A5 75 C2 93 i.e. upper limits on all motor motions. Execute, press any key and see F1=11110001 on the display indicating D3=D2=D1=0 i.e. limits were reached on steer CW (93), arm up (86), and head CW (C2), respectively. Press any key and then 3,2 to re-initialize.

Finally, replace the bytes at 0801-0807 by 00,00,4D,00,00,62,49 i.e. the initialization bytes themselves. Execute. No motion of any motor takes place, as is to be expected. Press any key and see DF=11011111 on the display. D5=0 indicates the limit on the arm (fully down) exists. No other limits are invoked in this case.

The next experiment presents an interesting robot safety application in which the limit switches are read and various alarms or pilot lights turned on if certain limits have been reached. In the program the provision is made for telling us which motors reached their limits (e.g. head and/or arm).

EXPERIMENT 2-11

POLLING LIMIT SWITCHES AT PORT C260 IN A ROBOT SAFETY APPLICATION. READ-OUT OF WHICH MOTOR LIMITS WERE REACHED. TURN-ON OF VARIOUS ALARMS OR PILOT LIGHTS.

In this practical robot application, we shall have the head turn a programmable amount followed by the raising of the arm a programmable amount. If neither, either, or both reach the maximum allowable limits (C2 for head CW and 86 for arm all the way up), 4 LEDs on our experimental board at outport C220 will indicate which ones, if any, reached their limits. You can regard these LEDs as pilot lights or different alarms corresponding to different motors. Thus, if neither reaches its allowable limit, our LEDs will show 0000. If the head reaches its limit C2, but the arm doesn't reach 86, the LEDs will show 0000 until the head limit is reached, then they will display 0001. If the arm reaches its limit, but not the head, the LEDs will show 0000 till the arm limit is reached, then change to

0010. If the head, then the arm reach their limits the LEDs will display 0000 till the head reaches its limit at which time they'll change to 0001 until the arm reaches its limit when they'll change to 0011. In other words bit 0 at the LEDs tells us whether the head reached its limit or not while bit 1 tells whether the arm reached its limit or not.

The program below is somewhat of a leap forward for us at this stage and will show the power we can achieve when mixing up R.L. and M.L. commands and Op Codes in the same program. It will also introduce the concepts of a "look-ahead" motor move interpreter command (R.L.) and a "branch if arm busy" R.L. command, both awfully powerful in general robot software control, aside from our vehicle HERO. The CC "motor move, continue, absolute (immediate)" command means that when the motor starts to move as a result of execution of this command by the monitor's interpreter, the program goes right on to the next instruction. It does not wait for motor move completion as a C3 "motor move, wait, absolute (immediate)" command would have it do. Thus the CC command has a "look-ahead" feature. The interpreter is weaving back and forth, in and out, continuing execution of the motor move and taking stabs at executing the following instructions. The format for the CC command is the same as for the C3 command: CC SS XX where SS determines which motor is invoked, and the speed it is to move at, while XX defines the absolute position the motor is to move to. SS and XX are arrived at in the manner explained before. (See Expt. 2-6, Appendix A "Programmer's Information Sheet", and Appendix B, where a useful table of SS bytes for various motors at various speeds (abs. or rel. moves) is presented based on SS = mmmssDdd and XX = ddddddddd-see Appendices A and B for definitions of m,s,D, and d). Thus SS=D0 for head motion at medium speed (abs.) and SS=50 for arm swing at medium speed (abs.)

We shall also employ the R.L. interpreter command 1E 00 which is defined as "branch if arm busy". 00 is the offset i.e. the relative address where we're to branch to and "arm" is defined with HERO as any one of head, shoulder (arm), extend, wrist rotate or pivot, or gripper motors. It is a powerful instruction (R.L. command) that allows us e.g. to poll the status of some parameter as long as the arm is busy moving. It is clear that 1E is a "natural" to be used after a CC command in any "look-ahead" polling scheme. Naturally, we shall be employing 3F (change to R.L.) and 83 (change to M.L.) frequently as we weave in and out between robot language commands and machine language Op Codes in the program that follows. We start at 07FB because we realized later that we had to clear the LEDs as an initialization. Remember: head limit (CW) is C2 and arm limit (up) is 86 on HERO.

INITIALIZATIONS, MOVE HEAD, POLL HEAD (CW) LIMIT SWITCH

07FB	86 00	LDAA 00
FD	B7 C2 20	STAA C220. Clear LEDs on experimental board.
0800	C6 00	LDAB 00. Initialize ACCB to 0. B will indicate if head limit reached or not.
02	3F	SWI. Change to R.L.
03	CC D0 XY	Move head medium speed to position XY (parameter) and continue on in the program as head moves. Later, choose XY=C2 (limit) or less.
06	83	Change back to M.L.
07	B6 C2 60	LDAA C260, Read limit switches at port C260.
0A	81 FD	CMPA FD. Compare ACCA with FD. Did head reach its CW limit C2? (See expt. 2-10 re:FD).

```
  0C      27 17       BEQ 17.  Yes, head CW limit (C2) reached, branch to 0825.
                      (BEQ:  branch if result = 0)
  0E      3F          Change to R.L.  C2 limit not reached.
  0F      1E F5       Branch if arm (head) busy back to 0806 to poll the limit
                      switches again.  Head still moving, XY≠C2 not reached.

HEAD NO LONGER MOVING, XY WAS REACHED.  MOVE ARM AND POLL ARM (UP) LIMIT SWITCH.

0811      3F          Change to R.L. Head reached XY, no longer busy.
  12      CC 50 UV    Move arm at medium speed to position UV and continue on in
                      the program.  UV is a parameter.  Later, choose UV=86
                      (limit) or less.
  15      83          Change to M.L.
  16      B6 C2 60    LDAA C260.  Poll limit switches.
  19      81 FB       CMPA FB.  Compare ACCA with FB.  Arm-up limit (byte 86)
                      reached?  (Head CW limit (C2) not reached).  (See Expt. 2-10
                      re: FB).
  1B      27 11       BEQ 11.  Yes, arm up limit (86) reached, head C2 limit not
                      reached, branch to 082E.
  1D      81 F9       CMPA F9.  Were both arm up limit (86) and head CW limit
                      (C2) reached?  (e.g. see 0825-2B).  (See Expt. 2-10 re: F9).
  1F      27 0D       Yes.  BEQ 0D.  Branch to 082E (arm up limit 86 and head
                      limit C2 reached).
  21      3F          Arm up limit (86) not reached.  Change to R.L.
  22      1E F1       Branch if arm (shoulder) busy (back to 0815 to poll the
                      limit switches again).
  24      3A          Arm reached XY≠86.  No longer busy.  Return to executive
                      ("ready").  Both head and arm have moved the requested
                      amounts XY,UV which are less than limits C2,86.

                HEAD LIMIT C2 REACHED.  TELL IT TO THE LEDs.

0825      86 01       LDAA 01
  27      16          TAB.  Save 01 in B.
  28      B7 C2 20    STAA C220.  See 0001 at LEDs.
  2B      7E 08 11    JMP 08 11.  Head limit C2 reached, 0001 shows at LEDs, start
                      shoulder (arm) swing up at 0811,12.

                ARM LIMIT 86 REACHED.  TELL IT TO THE LEDs.

082E      86 02       LDAA 02.
  30      1B          ABA.  Add B to A.  If B=00 (head limit not reached) then
                      we'll simply output 02 to LEDs at 0831.  If B=01 because
                      head limit was reached (see 0825-27) then ABA will cause 03
                      to be output to LEDs at 0831.  Thus 0010 at LEDs will tell
                      us arm limit only was reached while 0011 at LEDs will tell
                      us arm and head limits were reached.
  31      B7 C2 20    STAA C220.  Output 02 or 03 to LEDs at port C220.  See
                      either 0010 or 0011.
  34      3F          Change to R.L.
```

35	3A	Return to monitor ("ready"). Both head and arm have moved the amounts XY and UV, respectively. Arm has reached up limit (86) and head possibly reached its CW limit (C2) also.

There are four cases to be tried:

XY at 0805 (Final Head Position)	UV at 0814 (Final Arm Position)	Expected LED Display
72	20	0000
C2*	20	0000 then 0001
72	86**	0000 then 0010
C2*	86**	0000 then 0001 then 0011

* CW head limit; ** up arm limit.

Try these four cases and observe the LEDs.

Limit switches and reading their on/off status are very important in robot safety considerations. Being able to tell which motors (degrees of freedom) want to exceed the safe limits of the robot's work space at any time during the robot's activities ("on the fly") is a very critical matter. The above program and experiment indicates how this can be done.

EXPERIMENT 2-12

USING OUTPORT C2A0 TO CONTROL MAIN DRIVE (ROBOT'S BASE): FORWARD/REVERSE, ON/OFF, SPEED.

When the CPU outputs a byte to outport C2A0 via a B7 C2 A0 (STAA) instruction, the latched bits D7...D0 at port C2A0 drive the "Main Drive Circuit Board" (see "Block/Interconnect Diagram" that comes with your technical manual). These bits control the following main drive parameters:

D7: Forward or reverse motion (0 or 1, respectively).
D6: Main drive power on or off (1 or 0, respectively).
D5...D0: Speed of motion (111111 = max. speed, 000000 = at rest).

Consider the following program to control the main drive:

```
0700 86 XY     LDAA XY (XY=control parameter).
  02 B7 C2 A0  STAA C2A0.  XY to port C2 A0.
  05 BD F7 77  JSR INCH.  Waits for key press.
  08 86 0F     LDAA 0F to cut drive (D6=0).
  0A B7 C2 A0  STAA C2A0.  Main drive off.
  0D 20 FE     BRA FE.  Halt.
```

Consider the following XY bytes:

XY=C1: Reverse motion at very slow speed.
XY=DF: Reverse motion at medium speed.
XY=FF: Reverse motion at very fast speed.

Put the above XY bytes in the program at 0701 and in each case run the program. See reverse motion at the speed indicated above. Press any key to stop motion (cuts drive). Motion stops because 86 0F at 0708 together with B7 C2 A0 at 080A causes D6 to be 0 (D6 is the power on/off bit) cutting off drive power. 86 BF at 0708 will work equally well (why?).

Now consider the following XY bytes:

XY=41: Forward motion at very slow speed.
XY=5F: Forward motion at medium speed.
XY=7F: Forward motion at very fast speed.

Put the above bytes at 0701 in the program and execute for each case. Observe the forward motion at the expected speeds. Press any key to abort motion. Will 86 BF at 0708 also work?

For some fun and games you can program to have the robot go forward, then press any key to have it go back, then press any key to have it go forward again, etc. A program that accomplishes this follows:

```
0700    86 41       LDAA 41.
  02    B7 C2 A0    STAA C2A0.  Goes forward at very slow speed.
  05    BD F7 77    JSR INCH.  Waits till key pressed.
  08    86 C1       LDAA C1.
  0A    B7 C2 A0    STAA C2A0.  Goes reverse at very slow speed.
  0D    BD F7 77    JSR INCH.  Waits till key pressed.
  10    20 EE       BRA EE back to 0700.
```

Execute. See slow forward motion. Press any key and see robot go in reverse (slow). Press any key and robot reverses to go forward again (slowly), etc.

Finally, if at 0710 in the above program 20 EE is replaced with 86 0F followed by B7 C2 A0 and then 20 FE, the robot will go forward, reverse on the press of any key, and then stop when another key is pressed. Again 86 BF will work equally well.

Thus port C2A0 controls direction of drive, whether on or off, and its speed if on.

EXPERIMENT 2-13

THE REMOTE TEACH PENDANT SETTINGS IN (MANUAL MODE CONTROL) AND THE "SLEEP/NORMAL" SWITCH AS INPUTS AT INPORT C280.

You are urged to study inport C280 on your "Block/Interconnect Diagram" and the schematic of the teaching pendant (remote) circuit board on your "part 1 of 4" schematic (595-2872: ET-18). Port C280 is the input port holding various remote teaching pendant settings (switch bits) as well as a switch setting called "Sleep/Normal" (0/1). The latter is input line (bit) D1 to C280 and emanates from just above the experimental circuit board on the top of HERO's head. Before dealing with the remote teaching pendant, we'll perform an experiment showing the properties of the "sleep/normal" switch and how it can be used. Remember, when this switch is in the hi position (D1=1) it is in the "normal" state; when in the lo position (D1=0) it is in the "sleep" state. The difference between these states is simply this: the "sleep" state turns off all power except to RAM memory so that programs can be saved and HERO battery power conserved. HERO can stay in this state for several days before battery power starts to go low. In this state the display is naturally "dead", except for occasional flicks on the display to let you know it's in the sleep state. The normal state allows normal power to all HERO components so that it can carry out all the robot activities that you program for. The experiment dealing with the "sleep/normal" switch (D1) input to port C280 follows:

```
0600      B6 C2 80      LDAA C280.  Read port C280.
  03      85 02         BITA: ANDs (ACCA) with 02.  This tests state of D1 ("sleep/
                        normal").
  05      27 F9         BEQ F9.  If D1=0 ("sleep" state), branch back to 0600 and
                        read C280 again.
  07      BD F6 4E      JSR REDIS.  Fall through.  Switch was in "normal" (D1=1).
  0A      BD F7 77      JSR INCH.  Enter a key.
  0D      BD F7 AD      JSR OUTBYT.  Displays code of key entered.
  10      20 EE         BRA EE.  Branch back to 0600 and poll D1 (sleep/normal line)
                        into C280 again.
```

With the sleep/normal switch in the "sleep" state (D1=0 at C280), run. Press various keys. Nothing happens on the display. Now throw switch to the "normal" state (while the program is still running). Pressing keys will now show their codes on the display. Again throw the switch into the "sleep" state. Further striking of keys on the keyboard will have no effect on the display. Again throw the switch to the "normal" state. Entering keys will again show their codes on the display, etc.

We now turn our attention to the Remote Teaching Pendant's inputs to inport C280. A glance at your "Block/Interconnect/Diagram" shows that every setting of the pendant: Function (Arm/Body); Motion (Left/Right); Head; Gripper; Speed (Wrist Pivot, Wrist Rotate; or Arm Pivot, Arm Extend); Direction (Forward/ Reverse); Sleep/Normal Switch; Trigger (Press/Release) should produce a **unique** byte (code) at inport C280 according to the individual bits each individual setting produces. We can infer these various codes for various pendant settings by reading ports C280 into the CPU and then outputting same to the display. A program to do this follows. Please note that there is a utility resource in HERO's monitor which can be accessed upon Reset by simply pressing key 4 ("Manual Mode" - meaning we wish to implement remote manual control with the pendant). This "manual mode" utility must certainly have a part of its program which is equivalent to ours below, in which the monitor can identify, as do we in our program, the code for the particular pendant setting so that it can produce the required robot action being requested by that particular manual setting on the pendant. More about this later in Chapter 7 which is devoted entirely to the teach pendant. There, we not only implement remote manual control but also teach the robot various actions as dictated by various pendant settings, which actions the robot can then reproduce from memory later.

The program to decipher the codes corresponding to various pendant settings follows (connect pendant to lower plug on HERO's rear below arm).

```
0300    BD F6 4E    JSR REDIS
  03    B6 C2 80    LDAA C280.  Read pendant and sleep/normal settings.
  06    BD F7 AD    JSR OUTBYT.  Show code of pendant setting on the display.
  09    CE 10 00    LDX 1000 for time delay.
  0C    09          DEX.  Start time delay.
  0D    26 FD       BNE FD
  0F    20 EF       BRA EF to 0300.  Poll for possible new pendant/sleep setting.
```

We shall read the pendant codes for the following three pendant settings:

a) Function: Arm; Rotary to Arm/Pivot; Motion: Right; Trigger: pressed.

b) Function: Arm; Rotary to Head; Motion: Left; Trigger: pressed.

c) Function: Body; Rotary to Wrist/Pivot; Trigger: pressed.

With the sleep/normal switch in "normal", execute the above program and observe following codes on the display corresponding to settings (a), (b), (c) above:

a) D7; b) FB; c) 1F.

Note that the above bytes change to D5,F9,1D, respectively, when readings are taken with the sleep switch in the "sleep" setting. This again shows that the D1 line to C280 is 1 for "normal" setting and 0 for "sleep" setting of the "sleep/normal" switch, as our prior program demonstrated. Note also that, with sleep switch in normal setting and trigger released, the above bytes become D6,FA, 1E, respectively.

For a complete listing of all the codes corresponding to all pendant settings, you are referred to pp.58-61 of your technical manual. You can verify them all.

The fun begins and the light dawns when we Reset the microcomputer and press key 4 to enter the utility program in the monitor "manual mode" i.e. remote manual pendant control. If we then set the pendant to the exact settings specified in (a),(b),(c) above (always pressing the trigger last after the other settingsa are made), we find the following remote control robot actions take place:

a) Code D7: Arm swings up. Release your motion and trigger grip from the pendant and arm motion stops.

b) Code FB: Head rotates CCW. Release motion and trigger grip, head motion stops.

c) Code 1F: Body goes forward. Release trigger and motion and body motion ceases.

(Note in each case how the corresponding motor position is shown incrementing on the right-most two display digits; more on that in Chapter 7).

Now that we see what utility mode (Reset/key 4) does with the (a),(b),(c) pendant settings in remote manual operation, let's see if we can duplicate these actions by our own machine language program by polling port C280 for the pendant code corresponding to our pendant setting and then forcing the robot to take the appropriate action corresponding to that setting as observed in (a), (b), or (c) above.

Such a program of ours below must, by and large, be what utility mode 4 (manual remote control) is executing in HERO's monitor, when invoked, in order to produce the actions just observed. In the program below, remember C3 SS XX is the interpreter command for "motor move, wait, absolute, immediate" with SS=50 for arm swing (medium speed), or D0 for head rotate (medium speed) or 10 for forward body motion (medium speed); and XX=absolute position destination requested (see Appendix B).

```
0400    B6 C2 80      LDAA C280.  Poll pendant.
  03    81 D7         CMPA D7.  Pendant setting=D7(a)?
  05    27 0B         BEQ 0B.  Yes.  Branch to 0412 to swing arm up.
  07    81 FB         No.  CMPA FB.  Pendant setting=FB(b)?
  09    27 0F         BEQ 0F.  Yes.  Branch to 041A to rotate head CCW.
  0B    81 1F         No. CMPA 1F.  Pendant setting=1F(c)?
  0D    27 13         BEQ 13.  Yes.  Branch to 0422 to move body forward.
  0F    7E 04 00      JMP 0400.  None of the settings (a),(b),(c); poll again.

          PENDANT CODE WAS D7 (CASE (a)).  ARM SHOULD SWING UP.

0412    3F            Change to R.L.
  13    C3 50 30      Swing arm up to position 30.
  16    83            Change to M.L.
  17    7E 04 00      JMP 0400 and poll pendant again.

        PENDANT CODE WAS FB (CASE (b)).  HEAD SHOULD ROTATE CCW.

041A    3F            Change to R.L.
  1B    C3 D0 50      Rotate head CCW to 50.
  1E    83            Change to M.L.
  1F    7E 04 00      JMP 0400.  Poll pendant.

        PENDANT CODE WAS 1F (CASE (c)).  BODY SHOULD GO FORWARD.

0422    3F            Change to R.L.
  23    C3 10 08      Move body forward 08 units.
  26    83            Change to M.L.
  27    7E 04 00      JMP 0400 and poll pendant.
```

PROCEDURE

Execute the program with the teach pendant connected at the lower plug below the arm at HERO's rear. Nothing happens. Set the pendant to case (a) corresponding to code D7 and see arm swing up and then stop. Then set the pendant for case (b) (code FB) and see head rotate CCW and stop. Next set the pendant for case (c) (code 1F) and see HERO move forward 08 units and stop. These are, in general terms, exactly what HERO did when under control of utility mode 4 (manual remote) with the pendant connected and set for cases (a),(b),(c) above. (Always release trigger as soon as motion starts). You can do the above in any order of pendant settings e.g. (c), (a),(b), etc.

Now Reset and change 30 at 0415 to 00, 50 at 041D to 62, and 10 at 0424 to 14 (reverse body motion - Appendix B). Re-run and repeat the above procedure for each pendant setting (a),(b), or (c) and see motors go back to their initial states.

This then is how the remote manual pendant control (and later on, in its employment as a teaching pendant) works and how the monitor in utility mode 4 (Reset, key 4) recognizes pendant settings and initiates robot actions accordingly. We have accomplished here with our own machine language program what the monitor must be doing when in utility mode "4". This should remove some of the mystery as to how the pendant works. See Chapter 7 for much more on the remote pendant.

EXPERIMENT 2-14

USING OUTPORTS C2C0, C280, AND C260 TO GAIN DIRECT ACCESS (IN MACHINE CODE) TO HERO'S 7 STEPPERS: EXTEND, SHOULDER, WRIST PIVOT/ROTATE, GRIPPER, HEAD, STEER. DRIVING THE STEPPERS AT ANY SPEED AND COUNTING STEPS.

PRELIMINARY INFORMATION

a) D7....D0 Pin Assignments on Outport C2C0, in Particular Arm Select Pins D7,D6:

A glance at Appendix A and your technical manual's "Block/Interconnect Diagram" shows that D7 and D6 at outport C2C0 ("arm select A" and "arm select B") drive "enable A" and "enable B" of the "Arm Drive Circuit Board". They are the only two bits of C2C0 that interest us here. We have already worked with bit D5 of C2C0 (speech strobe for the speech circuit board). A 1 on D7 enables "arm select A", a 0 disables it. A 1/0 on D6 enables/disables "arm select B". (D4 is a reserved (spare) line and lines D3,D2,D1,D0 drive the time clock). Arm select A, when enabled, can choose any one of the wrist rotate, shoulder, or head steppers, and arm select B, if enabled, can choose any one of the wrist privot, gripper, or extend/retract steppers. Which is chosen from the group depends on the discussion in (b) below.

b) D7....D0 Pin Assignments on Outports C280 and C260 Driving Hero's 7 Steppers:

Again referring to the I/O schematics in our Appendix A and to the "Block/ Interconnect Diagram" in your technical manual, we find the following D7....D0 pin assignments at outports C280 and C260.

Outport C280

D7,D6,D5,D4 at port C280 are the four bits that drive either the extend/retract or head rotate steppers on the Arm Drive Circuit Board, depending on whether arm select B or arm select A is enabled (1), respectively. (See section (c) below on explanation on how the 4 bits do the driving).

D3,D2,D1,D0 at port C280 are the four bits that drive either the shoulder or the gripper steppers on the Arm Drive Circuit Board, depending on whether arm select A or arm select B is enabled (1), respectively. (See section (c) below on stepper 4 bit drive).

Outport C260

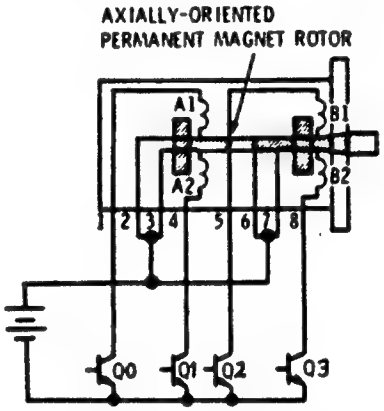
D7,D6,D5,D4 at port C260 are the four bits that drive the steering stepper on the Main Drive Circuit Board. (See section (c) below).

D3,D2,D1,D0 at port C260 are the four bits that drive either the wrist pivot or wrist rotate steppers on the Arm Drive Circuit Board, depending on whether arm select B or arm select A is enabled (1), respectively. (See section (c) below).

c) Bit Patterns Required at Outports C280 or C260 to Produce Individual Steps at any Stepper Motor:

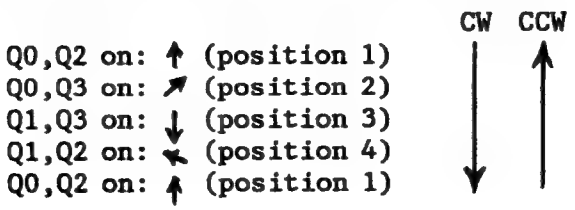
The circuit for a unipolar 2 phase stepper motor is shown below.

(Courtesy Heath Co.)



In the above circuit the base of each transistor Q3,Q2,Q1,Q0 is driven from the corresponding bit D3,D2,D1,D0 (or D7,D6,D5,D4, depending on the situation) coming from either port C280 (upper or lower nibble) or port C260 (upper or lower nibble) described above. They arrive from port C280 or C260 only after an instruction in your program first loads ACCA with the desired upper or lower nibble to drive the desired stepper (more on that below), followed by an instruction that outputs that nibble to data bus with latching to either port C280 or C260. Thus 86 XY followed by B7 C2 80 or B7 C2 60 will provide the necessary stepper drive bits. (Please remember, we are here going to drive the steppers by raw, fundamental machine language. After that we'll always drive steppers the easy way - by using HERO's robot language (R.L.) in our programs). A 1 coming from Di to the base of Qi in the above figure will cause that transistor Qi (and hence the stator coil in its circuit) to conduct giving rise to a magnetic field from that coil. Various combinations of magnetic fields (or absence thereof) from the various coils will cause the stepper's rotor to rotate to some definite equilibrium position determined by that combination. If we sequence those combinations in a definite order we can have the stepper rotate in steps CW or CCW.

The various rotor positions of the stepper for various Q3,Q2,Q1,Q0 "coil on" combinations are given below (shown by the arrow orientation):



From the above, we see that the bit pattern sequence required to produce individual steps in a CW or CCW fashion is as shown in the following table. Remember, Q3,Q2,Q1, Q0 are driven by the 1s or 0s on the four data bus lines D3,D2,D1,D0 (e.g. to drive wrist rotate stepper from port C260) or by the 1s or 0s on the four data bus lines D7,D6,D5,D4 (e.g. to drive the head rotate stepper from port C280).

BIT PATTERNS TO DRIVE A STEPPER

CW	CCW	Q3 D3 (D7)	Q2 D2 (D6)	Q1 D1 (D5)	Q0 D0 (D4)	Byte	Rotor Position
		0(off)	1(on)	0(off)	1(on)	05 or 50	↑
		1(on)	0(off)	0(off)	1(on)	09 or 90	↗
		1(on)	0(off)	1(on)	0(off)	0A or A0	↓
		0(off)	1(on)	1(on)	0(off)	06 or 60	↙
		0(off)	1(on)	0(off)	1(on)	05 or 50	↑

A PROGRAM TO CONTROL A STEPPER MOTOR ON HERO (MACHINE LANGUAGE). PROGRAMMABLE STEPPING TIMES. COUNTING OF THE STEPS.

In the program which follows, we shall first step the selected motor slowly so that each step of the motor in question can be heard, "felt", and seen. Later in the experiment we'll speed up the appropriate time delay bytes. Further, as we'll see, by changing a few parameters (port addresses or arm select lines/bits) we can change the stepper motor we wish to control. Thus the program is quite general in its scope.

We shall show the stepper bytes 05,09,0A,06 (or 50,90,A0,60) on the left-most two digits of the display and the number of steps made up to any given moment on the right-most two digits of the display (register B will store the instantaneous value of those steps). We shall also have the motor stop when it has made a desired (programmable) number of steps.

Remember the important points brought out in sections (a),(b),(c) above, which we summarize here:

1. To enable arm select B (wrist pivot, gripper, or extend/retract stepper) we'll need the initialization 86 40 (LDAA 40) followed by B7 C2 C0 (STAA C2C0). To enable arm select A (wrist rotate, shoulder, or head stepper) replace the 40 above by 80. 40 makes D6=1 at C2C0,80 makes D7=1 there.

2. To step any stepper CW we need the ACCA load instruction sequence 86 05 (50), 86 09 (90), 86 0A (A0), 86 06 (60) (LDAA) interspersed with outputs to the stepper in question, and in that order. Thus B7 C2 80 (STAA) would be the output instruction if shoulder, gripper, or head was selected vs. B7 C2 60 if wrist pivot, wrist rotate, or steer were selected, depending on which group (arm enable) was selected in (1) above. Note that the steer stepper does not require an arm select initialization as in (1) above. For CCW stepper motion the same remarks hold as above only the LDAA sequence would be 86 06 (60), 86 0A (A0), 86 09 (90), 86 05 (50).

Our program below will step the gripper (opening or closing it). Simple modifications given later will allow us to step the wrist rotate stepper, or the head rotate stepper, or any one of the 7 steppers, by simply changing a few bytes in the program.

```
0050    C6 00        LDAB 00.  B will count number of steps.
  52    86 40*       LDAA 40.
  54    B7 C2 C0     STAA C2C0.  Arm select B enabled i.e. pivot, gripper, or ex-
                     tend steppers.
  57    86 05*       LDAA 05.  Step byte for first CW position.
  59    B7 C2 80*    STAA C280.  05 to C280 will drive gripper to first CW step
                     position.
  5C    BD 00 80     JSR 0080 to "time delay and display stepper status" routine.
  5F    86 09*       LDAA 09.  Step byte for 2nd CW position.
  61    B7 C2 80*    STAA C280.  09 to C280 drives gripper to 2nd CW step position.
  64    BD 00 80     JSR 0080 to "T.D./display stepper status" routine.
  67    86 0A*       LDAA 0A.  Step byte for 3rd CW position.
  69    B7 C2 80*    STAA C280.  0A to C280 drives gripper to 3rd CW step position.
  6C    BD 00 80     JSR 0080 "time delay/stepper status" routine.
  6F    86 06*       LDAA 06.  Step byte for 4th CW position.
  71    B7 C2 80*    STAA C280.  06 to C280 drives gripper to 4th CW step position.
  74    BD 00 80     JSR to "T.D./display status" routine.
  77    C1 XY*,**    CMPB XY.  Compare (B) with XY.  XY=number of HEX steps to be
                     taken.  See Note below.
  79    27 FE        BEQ FE.  XY steps taken.  Halt.
  7B    7E 00 57     JMP 0057.  XY not reached.  Jump to 0057, continue stepping.
```

NOTES

* Bytes with * appended indicate they are parameters we can alter. Thus the 05,09, 0A,06 sequence can be changed to 50,90,A0,60 to invoke the head, extend, or steering steppers. Further, the CW sequence 05(50),09(90),0A(A0), 06(60) can be changed to 06(60),0A(A0),09(90),05(50) to achieve a CCW sequence. To select arm A, change 40 at 0053 to 80. To control steppers at outport C260 (see section (b)), change C280 to C260. XY determines the desired number of steps.

** Our program has a weakness in it: as written, XY must be taken as a multple of 4 since four increments of B at 0093 in the time delay below are executed before the CMPB at 0077 is made. This can be remedied by inserting C1 XY three more times after 005C,0064, and 006C in the program, thus comparing (B) with XY each time B is incremented.

TIME DELAY AND DISPLAY OF STEPPER STATUS

```
0080    CE C0* 00    LDX C000 (will give about 1/3 sec. time delay).
  83    09           DEX to achieve time delay.
  84    26 FD        BNE FD to 0083.
  86    BD F6 4E     JSR REDIS (left display digit)
  89    BD F7 AD     JSR OUTBYT.  Display (ACCA) (05,09,0A, or 06) on left 2 dis-
                     play digits.
  8C    4F           CLR ACCA.
  8D    BD F7 C8     JSR OUTCH.  Blanks 3rd display LED.
  90    BD F7 C8     JSR OUTCH.  Blanks 4th display LED.
  93    5C           INCB.  Increment ACCB.  One more step made.
  94    17           TBA.  Transfer B to ACCA.
  95    BD F7 AD     JSR OUTBYT.  Display number of HEX steps on right two LEDs.
  98    39           RTS.  Return from subroutine.
```

NOTE

* C0 can be changed at will to achieve larger or smaller time delays.

PROCEDURE

1. Run in machine language mode (i.e. Reset, key 1, key D,0050) with XY taken as 80H at 0078. In that case the gripper should stop after making 80H (128 dec.) steps. You'll observe the gripper open gradually in approximately 1/3 sec. time increments. The display will show, for example,....09-32; 0A-33; 06-34; 05-35;.... that is, the step driving byte (Q3,Q2,Q1,Q0) vs. the number of HEX steps made to that point. When the number of steps reaches (in this case) 80H, you'll see 06-80 on the display and the gripper stops stepping with the display "frozen".

2. Now insert 06,0A,09,05 at 0058,60,68,70 and execute the program again. This time you'll see the gripper step slowly in the other (closing) direction till 80H steps are made to a fully closed position. It stops there and the display will read 05-80.

3. Repeat (1) and (2) above with time bytes FF, then 20, then 08 at 0081 and notice the slower, and then faster, and then very much faster stepping action of the gripper.

4. Now control the wrist rotate stepper. To do this change 40 to 80 at 0053 to select arm A (wrist rotate/shoulder/head option) and put back 05,09,0A,06 at 0058,60,68,70. Change 80 at 005B,63,6B,73 to 60. Port C260 is thus invoked to involve the wrist rotate stepper because arm select A is now enabled. Change XY at 0078 from 80H to A0H (160 dec.). Finally, restore the time delay byte at 0081 to C0. Run this modified program and observe the wrist rotate in about 1/3 sec. time increments. The display again shows step drive bytes 05,09,0A,06, 05...and corresponding step count for each drive byte. The rotation stops with the display showing 06-A0 (A0 at 0078 will cause the CMPB and BEQ instructions at 0077 and 0079 to force a halt after A0H steps).

5. Now change the bytes at 0058,60,68,70 to 06,0A,09,05 and C0 at 0081 to 08, or even 01. Re-run and observe the stepper (for wrist rotate) rotate back in very fast steps, with the driving bytes and step number again showing on the display. The wrist stops rotating when the display shows 05-A0.

6. We shall now select the head rotate stepper motor. We do this by placing 80 at 0053 enabling arm select A (rotate, shoulder, or head option) and 80 back at 005B,63,6B,73. Port C280 is thus invoked and will involve the head rotate when arm select A is enabled. But since the head stepper at C280 is driven by D7,D6,D5,D4 we must use 50,90,A0,60 at 0058,60,68,70 (CW motion sequence). Let us choose F0 for XY at 0078 (F0H, or 240 dec, head steps). Restore C0 at 0081 as the time delay byte (slow). Execute and see the head step CW in about 1/3 sec. time increments a total of F0H steps. It then stops with the display showing 60-F0. While the head was stepping CW the display should show 50-01; 90-02; A0-03; 60-04; 50-05; 90-06, etc. You can actually hear a hollow thumping sound every time the head moves another step.

7. Replace 50,90,A0,60 at 0058,60,68,70 by 60,A0,90,50. Run and see the head turn CCW F0H steps till it gets back to its original position. The display will then show 50-F0.

8. Re-do (6) and (7) above twice; once with 20 at 0081 and then with 08 (or 01) at 0081 to observe fast, and then very fast stepping action. These cases give smooth head motion.

9. As a final case, let's control the steering stepper. Arm selects A and B have nothing to do with the steer motor. Hence the instructions at 0052 and 0054 are immaterial. Leave them in as is. The steer stepper is driven by D7,D6,D5, D4 at C260. Hence we must replace 80 at 005B,63,6B,73 by 60. The drive bytes at 0058,60,68,70 must now be 50,90,A0,60 for CW motion or 60,A0, 90,50 for CCW motion. Take XY at 0078 to be F0 (F0H steps) and the delay byte at 0081 to be C0 (slow). Run and observe the steering step action. It will be more apparent if you lift the front wheel slightly off the floor and monitor the steering movement of the wheel with your hand. The display reaches 60-F0 when movement of the steer motor halts. Or you can push HERO before you run the program. Notice its steering angle of motion. Push HERO after you run the program and the stepper has stopped. Compare the new steering angle with the first one. The difference will be significant and is attributable to the execution of the program.

We feel this experiment is a most important one from the point of view of learning how a stepper works, how it can be programmably controlled with a microcomputer to step any number of steps at any speed either CW or CCW, and in particular, how you can get at any one of HERO's 7 steppers by direct machine language if you know the addresses of the ports that select and control them. You are encouraged to invoke those steppers that we did not work with in this experiment (extend/retract, shoulder, wrist pivot) by modifying the appropriate bytes in the program.

You can now appreciate what the R.L. (robot language) interpreter commands like C3,CC,D3,DC, etc. do for us in one "fell swoop". What they do for us is execute programs like the last one for each stepper, for CW or CCW cases, and at slow, medium, or fast speeds. That's a lot of machine language programming that we are spared. Where are those M.L. programs? In HERO's ROM interpreter which an R.L. command of the type C3 (or CC or D3 or DC) SS XX invokes. So, of course, from now one we'll use R.L. motor move instructions rather than go through the M.L. approach (the hard way). We did the M.L. approach this one time to gain learning experience and true insight into how steppers work and how they can be controlled by a microcomputer. The secrets of the interpreter program and how the C3,CC etc. R.L. interpreter commands invoke the relevant routines in the interpreter will be revealed in Chapter 4.

Meanwhile, several more I/O ports and subjects remain to be covered for a thorough understanding of HERO: interrupt port C200; the real time clock (outport C2C0 and in/out ports C300); HERO's user dedicated keys; HERO's outport C1YX (the display LEDs) and some other matters. These topics are covered in Chapter 3 that follows.

CHAPTER 3

HERO'S USER DEDICATED KEYS; SPECIAL RAM LOCATIONS; ITS INTERRUPT PORT; ITS DISPLAY PORTS: REAL TIME CLOCK PORTS.

The material presented in this chapter involves experiments on aspects of HERO not treated in Chapter 2. It is necessary to gain further understanding of how total control of HERO can be achieved. Some of the material will involve I/O ports not taken up in Chapter 2: the interrupt inport C200; real time clock in/out ports C300 and outport C2C0 (one nibble of the latter was studied in Expt. 2-14 and Expt. 2-7 of the last chapter); and the display (LED) ports C1YX (Y defines which LED and X which segment thereon). These I/O ports and other special topics indicated in the above title merit a separate chapter here.

EXPERIMENT 3-1

HERO'S USER DEDICATED KEYS: 9,C,F.

From a Reset condition, entering key 9,C, or F automatically jumps user to address 0030 or 0033 or 0036 in RAM. A utility program in the monitor responds this way to the press of those keys after Reset. The monitor then reserves three addresses at each of the above locations (0030,31,32; 0033,34,35; 0036,37, 38) for user to jump to some other address of his/her choosing where he/she can write his/her own dedicated program, one to be always associated with the key 9, or C, or F i.e. one that might involve specific robot tasks that can easily be invoked by merely pressing key 9,C, or F after Reset.

An experiment demonstrating the use of these user dedicated keys is given below. The user, in this experiment, is to write the following jump (JMP) instructions at 0030-38:

```
Key 9:   0030     7E 08 00     JMP 0800
Key C:   0033     7E 08 20     JMP 0820
Key F:   0036     7E 08 40     JMP 0840
```

KEY 9 USER-DEDICATED PROGRAM. DISPLAY WILL SHOW 09 AND HEAD TURNS CW TO 82 THEN TURNS BACK TO 62.

```
0800      83*          Change to M.L.
  01      86 09        LDAA 09
```

```
  03      BD F6 4E      JSR REDIS
  06      BD F7 AD      JSR OUTBYT.  Show 09 on display
  09      3F            Change to R.L.
  0A      C3 D0 82      Move head to 82.
  0D      C3 D0 62      Rotate head back to 62.
  10      3A            Return to executive ("ready").  You can now press key 9 or
                        C or F again.

                KEY C USER-DEDICATED PROGRAM.  DISPLAY SHOWS 0C
                    AND ARM RISES TO 30 THEN LOWERS TO 00.

0820      83*           Change to M.L.
  21      86 0C
  23      BD F6 4E
  26      BD F7 AD
  29      3F
  2A      C3 50 30      Move arm up to 30.
  2D      C3 50 00      Move arm down to 00.
  30      3A            "Ready".  Press key 9,C, or F again.

          KEY F USER-DEDICATED PROGRAM.  DISPLAY SHOWS 0F AND
                WRIST ROTATES TO 6D THEN BACK TO 4D.

0840      83*           Change to M.L.
  41      86 0F
  43      BD F6 4E
  46      BD F7 AD
  49      3F
  4A      C3 70 6D      Rotate wrist to 6D.
  4D      C3 70 4D      Rotate wrist back to 4D.
  50      3A            "Ready".  Press key,9,C or F again.
```

*Each dedicated routine is seen to start with the R.L. command code 83 ("change to M.L."). This is necessary because the CPU arrives at 0800 or 0820 or 0840 from the jumps at 0030 or 0033 or 0036 in robot language (the monitor response to keys 9,C, or F does that for us). Since our programs at 0801,0821, and 0841 start in machine language, 83 is necessary "up front".

PROCEDURE

After entering the program press Reset, then enter keys 9,C,F in any order and as many times as you wish (always waiting till HERO says "ready" before pressing the next key so that the previous action can occur and terminate). You will see the dedicated robot motion associated with that key unfold and the byte 09,0C, or 0F accompany that action on the display.

EXPERIMENT 3-2

READING THE 7 STEPPER MOTOR POSITIONS FROM RAM BUFFER LOCATIONS 0000-0006 (AFTERWARD OR "ON THE FLY").

RAM locations 0000-0006 store, under monitor control, the updated "positions" of the steppers controlling "extend/retract", "shoulder (arm)", "wrist rotate", "wrist pivot", "gripper", "head", and "steering", respectively. These "positions" (HEX) are really number of steps made translated to some HEX number representing "position".

In the program that follows we drive the 7 motors to definite positions, then query the RAM buffer to see if RAM locations 0000-0006 confirm these attained positions. The program, as written, must be executed by pressing Reset, key A, key D, and 0700 (why?).

```
0700    FD 05 07    Motors, move all abs. (imm.) to 05,07,09,11,13,70,29
        09 11 13    (order: extend, shoulder, wrist rotate, wrist pivot, grip-
        70 29       per, head, steer)
  08    83          Change to M.L.
  09    BD F6 4E    JSR REDIS
  0C    CE 00 00    LDX 0000.  1st position at 0000 in RAM.
  0F    C6 03       LDAB 03 (3 positions to be read).
  11    BD F6 F9    JSR DISPLAY.  Will display contents of RAM locations 0000,
                    01,02.
  14    BD F7 77    JSR INCH.  Waits.  Press any key.
  17    BD F6 4E    JSR REDIS
  1A    CE 00 03    LDX 0003.  4th position at 0003.
  1D    C6 03       LDAB 03 (next 3 positions to be read).
  1F    BD F6 F9    JSR DISPLAY.  Will display contents of RAM locations 0003,
                    04,05.
  22    BD F7 77    JSR INCH.  Waits till you press a key.
  25    B6 00 06    LDAA 0006.  Read (0006), i.e. 7th position, into ACCA.
  28    BD F6 4E    JSR REDIS.
  2B    BD F7 AD    JSR OUTBYT.  See (0006) on the display (left-most digits).
  2E    BD F7 77    JSR INCH.  Press any key.
  31    3F          Change to R.L.
  32    3A          Back to monitor.  Says "ready"
```

PROCEDURE

Execute the program. See all motors move. Then see 05,07,09 on the display. Press any key and see 11,13, 70. Press any key again and see 29. Finally press any key and hear "ready". Now examine memory locations 0000 thru 0006. You should find 05, 07,09,11,13,70, 29. Reset and invoke the utility re-initialization program by pressing keys 3,2. Now examine RAM locations 0000-0006 and this time find 00,00,4D,00,00, 62, 49 (the initialized positions). Now change the bytes at 0701-0707 to 15,17,19, 21,23,80,39. Run and see 15,17,19. Press any key and see 21,23,80. Press any key and see 39. Press any key and hear "ready". Now examine memory locations 0000-0006 and find 15,17,19,21,23,80,39.

AN IMPORTANT PRACTICAL APPLICATION

An important variation of the above would be to have one motor run and, while running have its position displayed "on the fly". When that motor reaches the requested final position the next motor chosen runs and displays its updated position on the fly side by side with the fixed final position of the previous motor. Thus we shall be displaying instantaneous "on the fly" positions of the motors as they move.

The following program accomplishes this. We shall involve the head and shoulder motors and query RAM locations 0005 and 0001 as to their instantaneous positions.

```
0A00    CC D0 82     Move head to 82, continue on.
  03    83           Change to M.L.
  04    B6 00 05     LDAA 0005.  Read head position
  07    BD F6 4E     JSR REDIS
  0A    BD F7 AD     JSR OUTBYT.  Display head position.
  0D    3F           Change to R.L.
  0E    1E F3        Branch if arm (head) busy back to 0A03.
  10    CC 50 40     Head reached 82 (not busy)  Fall thru.  Move shoulder to
                     40.  Continue on.
  13    83           Change to M.L.
  14    BD F6 4E     JSR REDIS
  17    B6 00 05     LDAA 0005.  Read final head position (82).
  1A    BD F7 AD     JSR OUTBYT.  Display 82.
  1D    B6 00 01     LDAA 0001.  Read shoulder position
  20    BD F7 AD     JSR OUTBYT.  Show shoulder position next to final head
                     position on the display.
  23    3F           Change to R.L.
  24    1E ED        Branch if arm (shoulder) busy to 0A13.
  26    83           Change to M.L.  Arm reached 40.
  27    BD F7 77     JSR INCH.  Wait.  Press any key.
  2A    3F           Change to R.L.
  2B    3A           Back to monitor ("ready").
```

PROCEDURE

Execute (Reset, key A, key D, 0A00). See head turn CW toward 82. While it's doing so the display counts quickly from 62 (initial head position) to 82. Head then stops and arm starts to move. As it does the next two digits on the display count up from 00 to 40 (final arm position), all the while 82 is showing on the first two digits. Motion stops with the display reading 82,40. Press any key and hear "ready". Re-initialize by pressing keys 3,2. Change 82 and 40 in the program to 42,60 for final head/shoulder positions. Re-run. See the head, then shoulder, move and count-up of each one's position shown on the display. When motion ceases you see 42,60 on the display. Reset and examine RAM locations 0001 and 0005. You should read the bytes 60 and 42 there, respectively.

As a final note, replace CC D0 82 by DC D0 20 at 0A00-02. DC is the command code for "motor move, continue, rel. (immed)". Since the head starts from its initialized position 62, DC D0 20 is telling it to move a distance 20H units in the CW direction (see table Appendix B). This will also take the head to 82 just as CC D0 82 (abs) did. Run this modified program. You should see identical behavior on the display, showing that a relative motion command has no effect on the absolute stepper positions stored at 0005 (or any other of the locations 0000-0006).

EXPERIMENT 3-3

SPECIAL RAM LOCATION 0EE1.

RAM location 0EE1 is an interesting one even though we shall not use it much in our experiments. HERO's monitor uses it to store information telling us whether you ran your program in "Program Mode" (i.e. for a machine language execution in which you run by pressing keys Reset, 1,D, and starting address) or in "Repeat Mode". (i.e. for a robot language execution in which you run by pressing keys Reset, A,D and the starting address). To find out what the "semaphore" (flag) at 0EE1 is, to distinguish the 2 cases, simply run the following program first in "Program Mode", then in "Repeat Mode".

```
0100      B6 0E E1      LDAA 0EE1.  Read the semaphore.
  03      BD F6 4E      JSR REDIS.
  06      BD F7 AD      JSR OUTBYT.  Display semaphore byte on left two digits of
                        the display.
  09      20 FE         BRA FE (halt).
```

You'll find that 0EE1 reveals the byte 00 when execution is made in "Program Mode" and FF when it is run in "Repeat Mode". This can, at times, be very useful when we may want to execute a succeeding portion of program in robot language and may need a 3F to do so if 0EE1 reveals 00 or if we wish to execute a succeeding portion of program in machine language and may need the 83 R.L. Command if 0EE1 reveals FF. However, this is somewhat academic for us because 3F will convert to R.L. and 83 to M.L. in any, and all cases, period! In any case just be sure to run your program in "Program Mode" if it starts out in M.L. or in "Repeat Mode" if it starts out in R.L.

EXPERIMENT 3-4

USER POLLING OF INTERRUPT INPORT C200 FOR THE SOURCE OF 8 POSSIBLE INTERRUPTS. SERVICING SAME AT AN ADDRESS OF OUR CHOOSING.

Turn to the fold-out on p.119 of your technical manual (reproduced here in our Appendix A). Study the details of inport C200 there. It holds 8 possible sources of interrupts on its 8 tri-stated lines (see also p.10-61 of your Heath course notes also reproduced in our Appendix A). For a much more detailed study of the flip-flops (latches) and tri-states comprising inport C200 see the CPU circuit board schematic in your technical manual (595-2872, ET-18, part 2 of 4). For our purposes the general inport C200 schematic on P.119 fold-out of your technical manual will suffice. The input lines to C200 (held in tri-state status) are revealed there to have the following meanings:

D7: Possible interrupt from experimental board (INT).
D6: Possible interrupt from wheel pulse disc.
D5: Possible interrupt from trigger on remote pendant.
D4: Possible interrupt from the time clock.
D3: Possible interrupt from a "low voltage" situation on the logic board.

D2: Possible interrupt from a "low voltage" situation on the motor board.
D1: Possible interrupt from motion detection.
D0: Possible interrupt from the sonar (ultrasonic) system.

The above D7...D0 entities represent the situation on the input side of C200. In the diagram on p.119 of your technical manual (our Appendix A) you are reminded that on the output side of the C200 the above D7...D0 lines are connected to the data bus so that a reading of C200 (i.e. B6 C2 00) will input the byte D7...D0 into ACCA and hence give us a handle as to which interrupt line(s) is (are) high (1) and is (are) asking for an interrupt service request (a no-request situation is characterized by that line being lo(0)). Note also that in the diagram we are referring to, the D7...D0 outputs from C200 are not only wired to the D7...D0 lines of the data bus but are also input to a 7430 8-input NAND gate, so that when there is an interrupt request on any line, not only can we tell which line is requesting (by reading C200 into ACCA i.e. polling), but also the transition on the output side of the NAND gate will cause an interrupt in the execution of your program because it is wired into the IRQ (Interrupt Request) pin of the 6808, forcing a vectoring of the program counter to some interrupt service address (more on that later).

In this experiment we shall poll C200 for an interrupt from either "trigger" on the pendant or "INT" on the experimental board. In that case, we, the user, can then request exactly where we wish the INT (expt. board) or trigger (pendant) interrupts to be serviced i.e. we can vector to anywhere we wish for the servicing (we'll also deal with "motion detect" interrupts later in this experiment). That's because we're going to do the polling of C200. If we don't poll C200 but create an interrupt request on a particular interrupt source line, the monitor will poll for us (IRQ request will take us into the monitor in that case). If it finds a trigger interrupt it will jump us to 002A,2B, 2C in RAM (where user jumps himself to a convenient location for servicing), or to 002D,2E, 2F in the case of an interrupt from the experimental board (where again user can jump himself to where ever he likes for servicing), or to 0027,28, 29 in the case of a motion detect interrupt. We'll deal with the case of the monitor polling for the interrupt source (automatically done when IRQ makes a hi to lo transition), with its attendant vectoring to the definite addresses mentioned above, in Experiment 3-5 which follows. We'll go into great detail there on how the monitor does its polling of the interrupt source and just how it sends us to 0027 or 002A or 002D for a motion detect, trigger, or experimental board interrupt, respectively.

In order to inhibit the monitor from doing any polling if we are to successfully do the polling ourselves and vector ourselves to where we wish, we must make sure that the IRQ line, when it makes a request, is ignored (masked). That can be assured by employment of the SEI (0F), "Set the Interrupt Mask", instruction. Then when one of the interrupt lines makes a request our B6 C2 00 instruction in our program will poll as to which source line into C200 was responsible for the request without that request causing IRQ to vector us into the monitor for its interrupt polling routine (which would send us to one of the addresses mentioned above).

From the previous description of the interrupt lines D7...D0 into inport C200, we see that if an interrupt request comes from the experimental board (INT), the byte at port C200 would be 80, while it originates from "trigger" on the pendant, the byte at port C200 would be 20. Comfining ourselves for the moment to just these two possible sources of interrupts, the following program determines where the interrupt is coming from and jumps to 0910 if it's an experimental board interrupt or to 0930 if it's a trigger interrupt. The service routine we write at 0910 for

an experimental board interrupt will cause 80 to be displayed and the arm to move up and back down. The service routine we write at 0930 for a trigger interrupt will cause 20 to be displayed and the head to rotate CW and then back.

```
0900      0F           SEI.  Set the interrupt mask.  Disables the IRQ line so that
                       we will not be vectored into the monitor by a request from
                       one of port C200's sources.
  01      B6 C2 00     LDAA C200.  Read the interrupt port into ACCA.
  04      85 80        BITA 80.  Bit test.  AND (ACCA) with 80.  If result not zero
                       an experimental board interrupt exists, otherwise not.
  06      26 08        BNE 08.  Expt. board INT exists.  Branch to 0910.
  08      85 20        BITA 20.  Was not expt. board INT.  Was it trigger interrupt?
  0A      26 28        BNE 28.  Yes it was trigger interrupt.  Branch to 0930.
  0C      20 F2        BRA F2.  Was neither.  Branch back to 0900 and poll the in-
                       terrupt port again.

                EXPERIMENTAL BOARD INTERRUPT WAS MADE.  SHOW 80
                   ON THE DISPLAY AND MOVE ARM UP THEN DOWN.

0910      86 80        LDAA 80.
  12      BD F6 4E     JSR REDIS.
  15      BD F7 AD     JSR OUTBYT.  Show 80 on display.
  18      3F           Change to R.L.
  19      C3 50 20     Move arm up to 20.
  1C      C3 50 00     Move arm down to 00.
  1F      83           Change to M.L.
  20      BD F7 77     JSR INCH.  Press any key.
  23      BD F6 5B     JSR CLRDIS.  Clear display.
  26      7E 09 00     JMP 0900 to continue polling for more interrupts.

             TRIGGER (PENDANT) INTERRUPT WAS MADE.  SHOW 20
                ON THE DISPLAY AND MOVE HEAD CW AND BACK.

0930      86 20        LDAA 20
  32      BD F6 4E     JSR REDIS
  35      BD F7 AD     JSR OUTBYT.  Show 20.
  38      3F           Change to R.L.
  39      C3 D0 72     Move head to 72.
  3C      C3 D0 62     Move head back to 62
  3F      83           Change to M.L.
  40      BD F7 77     JSR INCH.  Press any key.
  43      BD F6 5B     Clear display.
  46      7E 09 00     JMP 0900.  Poll again.
```

PROCEDURE

1. Connect the pendant to HERO as a source of a trigger interrupt. A wire from INT on the experimental board will interrupt when you connect and disconnect that wire from one of the ground outlets available on the board.

2. Execute the above program. Create an interrupt from INT on the board and observe 80 on the display and see the arm go up and back down. Press any key. The display will blank. Now pull the trigger on the remote control pendant. See 20 on the display and observe the head move CW and back. Press any key when it stops

and see the display blank. Repeat these two interrupts in any order and see the above events repeat each time one or the other interrupt is made.

INCLUSION OF A MOTION DETECT INTERRUPT.

Patch in the R.L. command 4B ("enable motion detector") at 08FE followed by 83 (change to M.L.) at 08FF. Change 85 20 at 0908 to 85 02 (remember the motion detect interrupt is applied to line D1 of interrupt port C200). Change 26 28 at 090A to 26 48 to branch to 0950 where we write the routine servicing motion detection. Run from 08FE (Reset, A,D,08FE).

MOTION DETECT INTERRUPT WAS MADE. SHOW 02
ON THE DISPLAY AND SPEAK "MOTION".

```
0950     86 02        LDAA 02
  52     BD F6 4E     JSR REDIS
  55     BD F7 AD     JSR OUTBYT.  Show 02 on display
  58     3F           Change to R.L.
  59     72 09 60     Speak the phonemes starting at 0960.
  5C     83           Change to M.L.
  5D     7E 09 00     JMP 0900
  60     0C           0960-65 contain the phonemes for the word "motion".  3E at
  61     34           0966 gives a slight pause.  FF at 0967 forces a return to
  62     34           095C where (at 095D) a jump back to 0900 is made.
  63     11
  64     23
  65     0D
  66     3E
  67     FF
```

Execute from 08FE in Repeat Mode. Wave your hand at HERO. You'll see 02 on the display and hear the word "motion". Now cause an interrupt from INT on the experimental board. See 80 on the display and observe the robot's arm go up and down as before. Press any key to clear. Again wave at HERO and see 02 on the display and hear the word "motion" again, and again, and again for as long as you keep waving. Try another expt. board INT and again see 80 with arm motion up and back down, etc., etc.

We did not allow for polling of "trigger" when 85 20 at 0908 was changed to 85 02. To allow for such polling you'd have to add 85 20 and then the appropriate 26 XY BNE branch to your trigger routine. If this is done (and we suggest you try it) you'll be polling all three interrupt sources and all will work fine when any one of these interrupt sources inputs a request. Problem: do this suggested modification.

This experiment involved the three interrupt sources: "motion", "trigger", and "experimental board INT" at port C200. We learned how to poll those sources ourselves and when one was detected as the source of the interrupt, we were the ones to decide where the interrupt entry point (address) for servicing should be placed (see the BNE "branch if not equal to zero" instructions at 0906 and at 090A in the last program). However, if we do not inhibit the IRQ interrupt request from being made as a result of an interrupt request on one of the 8 inputs to C200 (remember, the outputs of C200 are NANDed together to form one input line to the IRQ pin of the 6808 microprocessor), then the IRQ request will cause the program counter to vector to a fixed address in the monitor. There the monitor can (and will) do the polling for us as to the nature of the source of the interrupt and, when it discovers that

source, it (the monitor) will send us to a service address of its choosing where we (user) must write the service routine. The IRQ request will be honored rather than inhibited if the interrupt system is enabled with a CLI (0E) "Clear Interrupt Mask" instruction in your program. All this is the subject of the next experiment.

EXPERIMENT 3-5

HERO'S MONITOR POLLS FOR THE INTERRUPT SOURCE AND JUMPS US TO 0027 (MOTION DETECT) OR 002A (TRIGGER DETECT) OR 002D (EXPERIMENTAL BOARD INT DETECT). EXAMPLES AND APPLICATIONS. WRITING SERVICE ROUTINES THERE.

We shall now allow the IRQ request (resulting from one or more source requests at C200-see p.119 fold-out of your technical manual) to be honored. This will force a program counter (PC) jump (vectoring) to a specific address in the monitor where the polling of the interrupt source takes place and the jumps to specific interrupt source addresses (for servicing) are settled for us by the monitor.

With the 6800 or 6808 microprocessor, an IRQ interrupt request causes a vectoring of the PC to address FFF8 (HERO or no HERO) - assuming the interrupt system is enabled (0E instruction). In HERO's monitor you will read the byte EF at address FFF8 and C9 at FFF9. The PC picks up these bytes EF and C9 and loads them into self so that the PC vectors once more to ROM monitor address EFC9. If you now read out the monitor instruction in HERO at EFC9 you'll find the very first instruction there is B6 C2 00. There you have it! The monitor is going to poll the source of the interrupt for us!

If you continue to read out the monitor's instructions from that point on you'll indeed see a polling of every possible interrupt source in the form of bit tests (BITA) in this order: 85 01, 85 40, 85 10, 85 08, 85 04, 85 02 (for "motion detect"), 85 20 (for "trigger detect"), and 85 80 (for "experimental board INT detect"). (Note that it does this polling with priorities: sonar first, wheel next, then time clock, then low voltage (logic board), then low voltage (motion board), then motion, then trigger, and finally experimental board). If you examine the monitor's instructions right after 85 02, you'll find 27(BEQ) 03 (motion interrupt not made) followed by BD 00 27 (motion interrupt was made). BD 00 27 stands for JSR 0027. Thus a motion interrupt will cause a vectoring to 0027 in RAM (monitor's doing). In the same way, 85 20 in the monitor is followed by 27 03 then by BD 00 2A (JSR to 002A), and 85 80 is followed by 27 03 then BD 00 2D (JSR to 002D). There it is! If a motion interrupt is detected the monitor jumps us to 0027; if a trigger interrupt is detected the monitor jumps us to 002A; and if an interrupt from experimental board INT is detected, the monitor jumps us to 002D (all on the assumption that the IRQ line was first enabled with an 0E (CLI) Op Code in your program). At each of these RAM entries the monitor leaves you 3 bytes to jump yourself (7E UV WX) to address UVWX of your choice in RAM where you can write the appropriate interrupt service routine.

Let's now perform an experiment which illustrates the use of these vector entry addresses as anchor points for jumps to other addresses of our choosing where we write service routines. The initial part of our program will itself write out the

7E UV WX jump instruction bytes to 0027,28,29;002A,2B,2C; and 002D,2E,2F. It will then go into a loop (simulating a "busy" program) waiting for an interrupt. We shall have our service subroutines display 02 and cause HERO to speak "motion" if a motion interrupt is made, or display 20 if a trigger interrupt is made, or display 80 if an experimental board INT is made. The program follows:

INITIALIZATIONS

```
0600    86 7E       LDAA 7E
  02    B7 00 27    STAA 0027.  7E written to 0027
  05    CE 06 30    LDX 0630.  Index register loaded with 0630.
  08    DF 28       STX 28.  06 to 0028, 30 to 0029 (direct addressing).
  0A    B7 00 2A    7E written to 002A.
  0D    CE 06 50    LDX 0650
  10    DF 2B       STX 2B.  06 to 002B, 50 to 002C.
  12    B7 00 2D    7E written to 002D
  15    CE 06 70    LDX 0670
  18    DF 2E       STX 2E.  06 to 002E,70 to 002F.
  1A    3F          Change to R.L.
  1B    4B          Enable motion detector.
  1C    83          Change to M.L.
  1D    0E          CLI.  Enable IRQ line.
  1E    20 FD       BRA FD.  Loop to 061D.  "Busy" loop waits for interrupt.
```

MOTION INTERRUPT DETECTED. DISPLAY 02 AND SPEAK "MOTION"

```
0630    3F          Change to R.L.
  31    5B          Disable motion detector (no motion "multiples" allowed)
  32    83          Change to M.L.
  33    0F          SEI.  Disable interrupt system.
  34    86 02       LDAA 02
  36    BD F6 4E    JSR REDIS
  39    BD F7 AD    JSR OUTBYT.  Show 02 on display.
  3C    3F          Change to R.L.
  3D    72 06 48    Speak the phonemes starting at 0648.
  40    83          Change to M.L.
  41    BD F7 77    JSR INCH.  Wait to press any key.
  44    BD F6 5B    JSR CLRDIS.  Clear display.
  47    39          RTS.  Return to loop at 061D-1F.
  48    0C          Phonemes for the word "motion".
  49    34
  4A    34
  4B    11
  4C    23
  4D    0D
  4E    3E          No sound.
  4F    FF          Back to 0640.
```

TRIGGER INTERRUPT DETECTED. DISPLAY 20.

```
0650    0F          SEI.  Disable interrupts.
  51    86 20       LDAA 20
  53    BD F6 4E    JSR REDIS
  56    BD F7 AD    JSR OUTBYT.  Show 20 on display.
```

```
  59      BD F7 77      JSR INCH.  Wait.  Press any key.
  5C      BD F6 5B      JSR CLRDIS.  Clear display.
  5F      39            RTS to loop at 061D-1F.
```

EXPERIMENTAL BOARD INTERRUPT DETECTED. DISPLAY 80.

```
0670      0F            SEI.
  71      86 80         LDAA 80
  73      BD F6 4E      JSR REDIS
  76      BD F7 AD      JSR OUTBYT
  79      BD F7 77      JSR INCH
  7C      BD F6 5B      JSR CLRDIS
  7F      39            RTS
```

PROCEDURE

1. First execute the program with 5B (disable motion detector) replacing 4B at 061B. That way any motion near HERO as we attempt to create trigger or experimental board interrupts will not create its own interrupt and mask out, or over-ride, our trigger or experimental board interrupts. A trigger interrupt will now show 20 on the display. Causing an interrupt from the experimental board should show 80 on the display. In each case press any key after the interrupt was made to clear the display and return to the loop at 061D to await another trigger or board interrupt.

2. Replace 5B with 4B at 061B. Wave at HERO. 02 will show on the display and HERO will say "motion". Press any key to clear the display. It won't work again. Why? Make it do so.

EXPERIMENT 3-6

IRQ INTERRUPTS AND THE STACK.

The significance of the following program will be evident in the discussion under the "procedure" section given after the program.

```
0700      8E 0E D0      LDS 0ED0.  Set stack pointer (SP) to 0ED0 in RAM.
  03      86 7E         LDAA 7E.
  05      B7 00 2D      STAA 002D.  7E to 002D.
  08      CE 07 20      LDX 0720.  0720 to X reg.
  0B      DF 2E         STX 2E.  07 to 002E,20 to 002F.
  0D      0E            CLI.  Enable IRQ line.
  0E      3E            WAI.  Wait for interrupt.
  0F      20 FE         BRA FE.  Halt.
```

EXPERIMENTAL BOARD INTERRUPT CAUSES IRQ. SERVICE ROUTINE.

```
0720      0F            SEI.  Disable IRQ line.
  21      86 80         LDAA 80.
  23      BD F7 AD      JSR OUTBYT.  See 80 on the display.
```

```
26      39          RTS.  Return to the "return address" 070F.
```

PROCEDURE AND DISCUSSION

1. Execute the program and cause an interrupt from the experimental board. See 80 on the display. Reset and examine stack locations 0ED0 and 0ECF. You should find 07 at 0ECF and 0F at 0ED0 - precisely the return address 070F. The interrupt was made with the processor in the WAI wait state at 070E. The address of the next instruction (the return address) is 070F. When the interrupt is made, this address is pushed on the stack, that is low address byte 0F is stored at 0ED0(SP) and high address byte 07 is stored at 0ECF(SP-1). Then when the 39 RTS instruction is executed at 0726 in the subroutine, the processor "pulls" the bytes in reverse order from the stack i.e. 07 is taken first from SP-1(0ECF), and placed in hi byte of PC and 0F is taken next from SP(0ED0) and placed in lo byte of PC, thus forcing the RTS to return to 070F, the correct return address. All is as it should be.

2. Now change 2D to 2A at 0707, 80 to 20 at 0722, and 2E to 2B at 070C. This will allow a trigger interrupt to be serviced at 0720 (via the 7E 07 20 this time at 002A,2B,2C). Now you'll see 20 on the display when you cause a trigger interrupt from the pendant. Reset and again examine the stack at 0ED0,0ECF. You'll find 07 at 0ECF and 0F at 0ED0, as it should be.

3. Re-do either (1) or (2) above with 8E 0E D0 at 0700 replaced by 8E 0E C0 Examine the stack for the return address at the proper locations in RAM for this case.

EXPERIMENT 3-7

THE 6 LED DISPLAY. I/O OUTPORTS C1YX.

Another I/O we should touch base with are the 6 digital LED segment displays above the keyboard. Their addresses are C1YX.

X=0,1,2,3,...7 for each segment in a given display as shown. A simple program to light up any segment in any display follows.

```
0500      86 01        LDAA 01.
  02      B7 C1 YX     STAA C1YX where Y=1,2,...6 and X can be 0,1,2,... or 7.
  05      20 FE        BRA FE.  Halt.
```

As an example, take YX=54. Run and see segment 4 of the second display from the left light up. Change YX to 36 and see segment 6 of the fourth display from the left light up. Next try YX=17 and see the decimal point to the right of the right-most display light up. Try other YX combinations.

EXPERIMENT 3-8

HERO'S REAL TIME CLOCK: OUTPORTS C2C0,C300 AND INPORT C300. APPLICATION TO READ-OUT OF TIME AND DELAYED ROBOT ACTION. A ROBOT "ON GUARD" DETECTING LIGHT OR SOUND.

The last of HERO's I/Os to be discussed and used will be outports C200,C300 and inport C300. They constitute HERO's Real Time Clock. It is most useful to be able to read this clock and display time. When time reaches a desired, programmable value you can then have the robot perform some desired action or have sensing take place with the action dependent on the sensed value. We shall give just such an application using HERO's real time clock.

You are urged to refer to outports C2C0,C300 and inport C300 (which constitute the real time clock) on p.35 of your technical manual (reproduced in our Appendix A here). The select inputs A3,A2,A1,A0 are most important and are related to the clock's output range and the variable selected (time unit) as follows (see also table on p.36 of your technical manual):

INPUT SELECT				VARIABLE SELECTED (UNITS OF ())	OUTPUT RANGE READ FROM CLOCK (LSN AT INPORT C300:D3,D2,D1,D0)
A3	A2	A1	A0		
0	0	0	0	Sec (1)	0-9
0	0	0	1	Sec (10)	0-5
0	0	1	0	Min (1)	0-9
0	0	1	1	Min (10)	0-5
0	1	0	0	Hrs (1)	0-9
0	1	0	1	Hrs (10)	0-1 (or 0-2)

We are omitting the rest of the table covering weeks, days, months, years (see p.36 your technical manual). Refer to this table and p.35 of your technical manual (or Appendix A here) when analyzing the program that follows.

In the following program the passage of time (in the units selected) will be shown on the display. The robot is inactive. When that time reaches a desired programmable value (UV at 011E in our program) the display "freezes". The robot then goes into action saying the phrase "on guard" and polling for light and sound. If it detects light it will say "I see light - on guard". If it hears sound it will say "I heard that - on guard". It will continue to do so every time light or sound is detected.

PASSAGE OF TIME DISPLAYED TILL DESIRED VALUE REACHED. ROBOT INACTIVE TILL THEN.

```
0100    86 A0       LDAA A0.  A0=10100000 will allow a read of the clock if it is
                    output to port C300.  See App. A.
  02    B7 C3 00    STAA C300.  Outputs a 1 to read line of clock and a 0 to
                    write line (read of clock requested).
  05    86 XY       LDAA XY.  XY will determine the variable selected e.g. XY=00
                    selects 1 sec. intervals, 01 selects 10 sec. intervals, 02
                    minutes, 03 10 minutes, 04 hours, etc.
  07    B7 C2 C0    STAA C2C0.  XY to C2C0 to realize the variable select.
  0A    B6 C3 00    LDAA C300.  Read the initial (arbitrary) time from C300.
  0D    16          TAB.  Save it in ACCB.
  0E    B6 C3 00    LDAA C300.  Read subsequent (relative) times into ACCA.
  11    11          CBA.  Compare A and B (A-B):  Subsequent relative time vs.
                    initial relative time.  Which is greater?
  12    24 02       BCC 02.  Branch if carry clear to 0116 if subsequent rela-
                    tive time is larger than the initial relative time.
  14    8B 0A       ADDA IMMED.  Subsequent relative time less than initial re-
                    lative time.  Make a correction:  add 0A to contents of ACCA
                    i.e. adding 10 dec. will give the correct subsequent rela-
                    tive time greater than initial relative time.
  16    10          SBA.  Subtract B from A.  A-B is placed in A.  ACCA now
                    holds the correct absolute elapsed time for either case i.e.
                    for the cases where subsequent read time is greater or less
                    than initial read time.
  17    BD F6 4E    JSR REDIS.
  1A    BD F7 B5    JSR OUTHEX.  Output one HEX character.  Will display abso-
                    lute elapsed time in the time quantity and units chosen by XY
                    at 0106
  1D    81 UV       CMPA UV.  Compare ACCA (time reached) with the desired time
                    limit UV.  UV=02 means 2 units of time is the limit, 03
                    means 3 units of time is the limit, etc., where units =1 or
                    10 sec., 1 or 10 min., 1 hr., etc. depending on value chosen
                    at 0106.
  1F    27 09       BEQ 09.  Time limit UV reached.  Branch to 012A where robot
                    action starts.
  21    CE 03 00    LDX 0300.  Time limit UV not reached, load 0300 into X
                    register to waste about 0.01 sec. before another reading of
                    time.
  24    09          DEX for small t.d.
  25    26 FD       BNE FD.  Do another DEX at 0124.  X≠0.
  27    7E 01 0E    JMP 010E.  Take another (updated) reading of relative
                    time on the clock C300.
```

TIME LIMIT UV REACHED. DISPLAY SHOWS UV LIMIT. ROBOT GOES INTO ACTION (SAYS "ON GUARD").

```
012A    3F          Change to R.L.
  2B    72 01 32    Speak the phonemes at 0132 ("on guard").
  2E    83          Change to M.L.
  2F    7E 01 40    JMP 0140 to poll light and sound sensors
  32    15          Phonemes for the phrase "on guard".
  33    23
```

```
  34      0D
  35      3E            No sound.
  36      1C
  37      15
  38      2B
  39      1E
  3A      3E            No sound.
  3B      FF            Back to 012E.
```

POLLING LIGHT AND SOUND SENSORS. IF LIGHT, ROBOT SPEAKS "I SEE LIGHT, ON GUARD"; IF SOUND, ROBOT SAYS "I HEARD THAT, ON GUARD".

```
0140      3F            R.L.
  41      41            Enable light detector
  42      83            M.L.
  43      B6 C2 40      LDAA C240.  Read light level.
  46      81 D0*        CMPA D0.  Does light level exceed D0?
  48      22 0C         BHI 0C.  Yes.  Branch to 0156.
  4A      3F            No.  Change to R.L.
  4B      42            Enable sound detector
  4C      83            M.L.
  4D      B6 C2 40      LDAA C240.  Read sound level.
  50      81 90*        CMPA 90.  Does sound level exceed 90?
  52      22 0A         BHI 0A.  Yes.  Branch to 015E.
  54      20 EA         BRA EA.  No light, no sound.  Poll again back at 0140.
```

LIGHT DETECTED. HERO SAYS "I SEE LIGHT-ON GUARD"

```
0156      3F            R.L.
  57      72 FB 37**    Speak the phonemes at FB37 in the monitor ("I see light").
  5A      83            M.L.
  5B      7E 01 2A      JMP 012A to say "on guard" again and then poll for light/
                        sound again.
```

SOUND DETECTED. HERO SAYS "I HEARD THAT - ON GUARD".

```
015E      3F            R.L.
  5F      72 FB 6B**    Speak the phonemes at FB6B in the monitor ("I heard that").
  62      83            M.L.
  63      7E 01 2A      JMP 012A to say "on guard" again and then poll for light/
                        sound again.
```

NOTES

* Choose light and sound level threshhold bytes to fit your ambiance.
** See pp.77,78 of your Heath Co. "Voice Dictionary" for a list of monitor addresses where "canned" phrases reside.

PROCEDURE

1. Take XY=00 at 0106 and UV=06 at 011E. Run. You'll see the count-up and display of time in 1 sec. intervals. When the count reaches 6 the count-up halts and the robot speaks the words "on guard". If now you talk, robot says "I heard that -

on guard". If you expose it to light it'll say "I see light - on guard". You can alternate between sound and light as many times as you like.

2. Change XY to 01 and UV to 03. Run. This time the display counts time in 10 sec. intervals. It will show 0,1,2,3 in 10 sec. intervals and stop when it displays the number 3. At that point HERO says "on guard". Again light and sound bring forth the words "I see light - on guard" and "I heard that -on guard", respectively.

3. Change XY and UV to other time units and limits.

CONCLUSION

Chapters 2 and 3 were designed to give you a total mastery of all of HERO's I/Os and its important memory buffers by means of a direct, machine language approach. You should by now have that mastery. With it, HERO will be in your "hip pocket".

CHAPTER 4

THE INTERPRETER BEHIND A ROBOT LANGUAGE. WRITING YOUR OWN INTERPRETER FOR YOUR ROBOT COMMANDS.

THEORY

The main purposes of this manual of experiments are knowledge and understanding (the fundamentals) and also applications in the real world. By now you must be very curious as to how HERO's interpreter really works to carry through HERO's "robot language" (R.L.) commands. When, for example, C3 ("motor move, wait abs (immediate)") is placed in the program, the latter looks upon C3 not as a 6800 Op Code that goes to the CPU for execution by the microprogram (no such 6800 code exists anyway), but as a HERO interpreter command to make a specified motor move to a specified position at a certain speed. Certainly the microprogram in the 6800 (6808) isn't going to do that for us - its job is to execute only 6800 (6808) Op Codes found on your 6800 instruction card.

Who then executes the command C3, and how does it do so?

You've noticed how we have used two "magic" codes, 3F and 83, from time to time in our programs. 3F precedes a stretch of HERO. R.L. (interpreter) commands if the preceding stretch was machine language code (M.L.). It allows the ensuing R.L. commands to be executed. 83, on the other hand, precedes a stretch of M.L. honest to goodness 6800 Op Codes if the stretch preceding 83 contained R.L. commands. Funny thing is that 3F is a real 6800 Op Code executed by the CPU's microprogram - it is not a robot command; whereas 83 itself is a HERO R.L. command not recognized by the CPU and must be executed by the interpreter.

In fact, 3F stands for "SWI" (Software Interrupt). What SWI does, when executed by the microprogram, is: 1) saves the address of the next instruction in the program (after 3F) on the stack, and 2) jumps to an address specified in locations FFFA and FFFB in memory (this is true for every 6800/6808, not just the one in HERO). At FFFA, FFFB the CPU's program counter picks up hi/lo bytes of an address (written there by the user) to which the program again jumps and there, at that address, user writes his SWI interrupt service subroutine to do what he pleases for whatever purpose. In the case of HERO, Heath Co.'s monitor writers placed 0F at FFFA and F6 at FFFB (check it yourself). Thus an SWI (3F) software interrupt causes a jump to FFFA,FFFB where the address 0FF6 is picked up vectoring us to that location in RAM.

If now you inspect location 0FF6 in RAM (after a Reset) you'll find another jump to F8AD in monitor ROM (i.e. 7E F8 AD codes at 0FF6,F7,F8). At F8AD and thereafter, a

machine language program (the "interpreter") for HERO has been written which interprets and implements every one of the R.L. interpreter commands that follows 3F in your program, including 83 which is itself a HERO interpreter command. As already stated, 83 will allow all codes following it in your program to be interpreted as 6800 M.L. Op Codes executable by the microprogram. (Hence you must be sure that after 3F in your program all codes are R.L. codes and that after 83 in your program all codes are genuine M.L. 6800 codes).

Thus there are two programs to be clearly delineated and kept in mind: the microprogram that executes 6800 Op Codes (M.L.); and the interpreter program (commencing at 0FF6 in HERO's RAM and going over to F8AD in its ROM) that executes and implements HERO's defined R.L. interpreter commands. (We should also mention here, for future reference in this chapter, that Reset is an interrupt (hard-wired) which, with all 6800/6808 CPUs, vectors to FFFE,FFFF. There the Reset subroutine address is found. With HERO you'll find F3 and 7E at FFFE,FFFF so that a Reset forces a vectoring to address F37E in HERO's ROM. There starts HERO's monitor ("executive") which is responsible for, among the many other things it does, the "HERO" message on the display and the word "ready" to be uttered).

The best way to understand how HERO's interpreter works to execute its R.L. defined commands (or how any other interpreter works, for that matter) is to define our own interpreter commands and write our own interpreter to execute and implement those commands. That should remove the mystery from how R.L. commands like C3,D3,E3,41,1E,83,72,4B,3A, etc. (in HERO) get executed.

To execute interpreter commands (HERO's or ours), the interpreter must use M.L. (6800 Op Codes) in its program to have I/Os like steppers (for example) do precisely what these commands specify. And that can only be done exactly the way we did it in Chapter 2 and 3 where we controlled all I/Os by machine language - the only language the CPU understands (see for example Expt. 2-14 where we drove and controlled HERO's 7 stepper motors). Thus a writer of an interpreter or any compiler for any higher level robot language must be able to have his interpreter or compiler execute desired robot actions (corresponding to defined commands) directly in the machine language of the CPU being used. This is done in the following experiment.

EXPERIMENT 4-1

WRITING OUR OWN INTERPRETER TO EXECUTE OUR OWN ROBOT LANGUAGE COMMANDS.

Let us now define our own "Robot Language" interpretter commands to be 3D,7B,93, and B3 with the following meanings:

1) 3D shall mean "turn on LEDs on the experimental board" (address C220).
2) 7B shall mean "sample and display (as a byte) the light intensity detected by HERO at the end of 1 second from the time the program executes".
3) 93 shall mean "change to machine language", (just as 83 does with HERO's interpreter).

4) B3 shall mean "return to executive and say 'ready'" (just as 3A does with HERO's interpreter).

None of these codes is a 6800 Op Code nor a HERO interpreter command. They are strictly new - ours.

Recall that an SWI (3F) software instruction, being an interrupt, stores the low byte of the next instruction's address (the return address) into memory location pointed to by SP (stack pointer) and the high byte of that return address into the memory location pointed to by SP-1. When that is done SP is decremented once more to a value SP-2. This is most important in the program that follows. For concreteness and program control we shall initialize SP to the address 0ED0 in RAM.

The program, as written below, will have a stretch of bytes 3F,3D,7B,93 which will mean, in our language, 1) change to R.L. (ours)-3F will do that, 2) turn on LEDs on the experimental board-3D will do that, 3) sample and display the light intensity one second after run time - 7B will do that, and 4) change back to M.L. so that the machine code following 93 in the program can again be executed by the CPU's micro-program-93 will do that.

It is suggested that you read very carefully all commentary in the program. Several sophisticated 6800 instructions involving stack pointer, index register, and indexed addressed reading into the accumulator are involved. They form the heart of our interpreter program. (Much the same is probably involved with HERO's interpreter at 0FF6/F8AD).

Finally note that 0FF6 is the "property" of HERO when 3F (SWI) is executed. That is, when you execute a program with 3F in it, you are taken to 0FF6 (via FFFA,FFFB) where HERO's jump instruction 7E F8 AD takes you to F8AD in HERO's ROM interpreter for the purpose of executing HERO's R.L. codes (alas 3D,7B,93,B3 are not among them). But we can overcome that and expropriate 0FF6,F7,F8 (the locations for the jump bytes) for ourselves. How? By simply having our program at the outset write the bytes 7E 06 00 into the locations 0FF6,F7, F8, respectively. That way the SWI (3F) code in our program, when it occurs, will vector us to 0FF6 where 7E 06 00 is picked up (7E:JMP) taking us to the address of our choice in RAM (0600 in our case). There we shall write our interpreter to implement our own R.L. interpreter commands 3D,7B,93,B3.

"Ready"? Here goes.

USER's INITIALIZATIONS

```
0500      86 7E        LDAA 7E.
  02      B7 0F F6     STAA 0FF6.  Write 7E to 0FF6.
  05      86 06        LDAA 06
  07      B7 0F F7     STAA 0FF7.  Write 06 to 0FF7.
  0A      86 00        LDA 00.
  0C      B7 0F F8     STAA 0FF8.  Write 00 to 0FF8.  This ensures that 0FF6, 7,8
                       contain "JMP 0600" when an SWI enters 0FF6 via FFFA,B.
  0F      8E 0E D0     LDS 0ED0.  Set SP to 0ED0.
```

USER'S MAIN PROGRAM

```
0512      86 FF        LDAA FF
```

```
14      BD F6 4E      REDIS
17      BD F7 AD      OUTBYT. Display FF.
1A      3F            SWI ("change to our robot language"). Software interrupt to
                      0FF6. Save return address 051B on stack (1B to 0ED0,05 to
                      0ECF with SP pointing to 0ECE after SWI executed). 7E 06 00
                      at 0FF6,7,8 will take us to 0600 below.
1B      3D            Our R.L. command to turn on the LEDs on the experimental
                      board.
1C      7B            Our R.L. command to read and display the light intensity de-
                      tected by HERO 1 sec. after the program is executed.
1D      93            Our R.L. command to change back to M.L. so as to execute the
                      machine code from 051E-052D.
1E      BD F7 77      JSR INCH. Waits for the press of a key.
21      86 00         LDAA 00
23      BD F6 4E      JSR REDIS
26      BD F7 AD      JSR OUTBYT. Display 00 when any key pressed.
29      B7 C2 20      STAA C220. Turn LEDs on expt. board off.
2C      20 FE         BRA FE. Halt.
```

OUR INTERPRETER: INVOKED WHEN SWI (3F) IS EXECUTED IN THE MAIN PROGRAM (0060 ENTERED VIA 0FF6 SWI ENTRY)

```
0600    8E 0E CE      LDS 0ECE. Make sure SP is where it should be (0ECE) and
                      where we want it after execution of 3F at 051A.
03      30            TSX. SP+1 to X. In this case, 0ECF goes to index register
                      but SP remains at 0ECE.
04      A6 00         LDAA X(00). Read contents of location X+00 (i.e. of loca-
                      tion 0ECF+00) into ACCA. In this case address 0ECF will
                      contain 05 (hi byte of return address 051B. Remember 1B
                      was saved at 0ED0 and 05 at 0ECF when 3F was executed).
                      Thus 05 is now in ACCA.
06      B7 07 00      STAA 0700. Save 05 in location 0700.
09      31            INS. Increment SP to 0ECF.
0A      30            TSX. SP+1(0ED0) to X. SP stays at 0ECF.
0B      A6 00         LDAA X(00). Read contents of location X+00(0ED0) into ACCA.
                      In this case address 0ED0 contains 1B (lo byte of return ad-
                      dress 051B). Hence 1B now is in ACCA.
0D      B7 07 01      STAA 0701. Save 1B in location 0701.
10      FE 07 00      LDX 0700. (0700)=05 goes to XH and (0701)=1B goes to XL.
                      Thus X now contains 051B.
13      A6 00         LDAA X(00). Reads contents of X+00 (i.e. of 051B, namely
                      our R.L. code 3D at 051B in the program) into ACCA for
                      polling.
15      81 B3         CMPA B3. Compare (ACCA) with B3. Was it our R.L. command
                      B3?
17      27 0C         BEQ 0C. Yes. Branch to 0625 to execute our R.L. command B3.
19      81 3D         CMPA 3D. No. Was it our R.L. command 3D?
1B      27 0B         BEQ 0B. Yes. Branch to 0628 to execute our R.L. command 3D.
1D      81 7B         CMPA 7B. No. Was it our R.L. command 7B?
1F      27 10         BEQ 10. Yes. Branch to 0631 to execute our R.L. command 7B.
21      81 93         CMPA 93. No. Was it our R.L. command 93?
23      27 2A         BEQ 2A. Yes. Branch to 064F to execute our R.L. command 93.
```

EXECUTE OUR INTERPRETER COMMAND B3
("RETURN TO MONITOR AND SAY "READY")

0625	7E F3 7E	JMP F37E. Jump to the same address F37E where the Reset key on HERO would take you if pressed (see THEORY section earlier in Chapter 4). See "HERO" displayed and hear "ready" spoken.

EXECUTE INTERPRETER COMMAND 3D
("TURN EXPERIMENTAL BOARD's LEDs ON")

0628	86 0F	LDAA 0F
2A	B7 C2 20	STAA C220. Write out 0F to LEDs on experimental board.
2D	08	INX. Increment the index register from its previous value 051B to 051C (see address 0610).
2E	7E 06 13	JMP 0613 where A6 00 will read contents of address 051C +00 (i.e. 7B) into ACCA for polling again.

EXECUTE INTERPRETER COMMAND 7B ("READ AND DISPLAY LIGHT VALUE ONE SEC. AFTER PROGRAM RUNS").

0631	86 B0	LDAA B0
33	B7 C2 E0	STAA C2E0. B0 to port C2E0 will enable the light detector, sense board and display (see Expts. 2-2,2-3, 2-4 Chapter 2).
36	FF 08 00	STX 0800. Store contents of index register into memory: XH=05 to 0800 and XL=1C to 0801. (We are going to use X in what follows).
39	CE FF FF	LDX FFFF. Will give about 1 sec T.D.
3C	09	DEX
3D	26 FD	BNE FD. X not yet 0.
3F	B6 C2 40	LDAA C240. One second elapsed. Read light value from sense board C240.
42	BD F6 4E	JSR REDIS
45	BD F7 AD	JSR OUTBYT. Display detected light byte.
48	FE 08 00	LDX 0800. (0800)=05 goes back to XH and (0801)=1C goes back to XL.
4B	08	INX index register from last value 051C to 051D (see address 062D).
4C	7E 06 13	JMP 0613 where A6 00 will read contents of address 051D+00 (i.e. 93) into ACCA for polling again.

EXECUTE INTERPRETER COMMAND 93
("CHANGE TO MACHINE LANGUAGE")

064F	08	INX. Increment X from last value 051D to 051E (see address 064B).
50	8E 0E D0	LDS 0ED0. Reset SP to 0ED0 where it was before the 3F (SWI) was executed at 051A.
53	6E 00	JMP X. Indexed addressing mode of JMP. Program will jump to the address given by (X)+00 i.e. in this case to 051E+00 which is precisely the address of the next instruction after 93 in the main program. There it picks up the stretch of machine code written by us at 051E and beyond. These codes are now executed by the CPU's microprogram as valid 6800 Op Codes.

PROCEDURE

1. Execute the above program. You'll see FF appear on the display and the LEDs on the experimental board turn on. About a second later FF on the display is replaced by the value of the light intensity byte detected at that time. Now press any key and see the light byte change to 00 and the LEDs on the experimental board turn off (the result of execution of machine code after 93 from 051E-052D).

2. Now change 7B at 051C to B3 ("return to monitor and say "ready"). Re-run and see the LEDs on the board turn on due to the 3D command at 051B. You'll then immediately see "HERO" on the display and hear the word "ready". B3 at 051C is our software Reset equivalent to HERO's 3A.

3. Or you replace 93 at 051D by B3 (putting 7B back at 051C). In that case execution will result in LEDs on the board going on and FF showing on the display for about 1 second. "HERO" is then displayed and the word "ready" pronounced. If you're lucky and possess a quick eye you'll "see" the light byte value quickly replace FF on the display just before B3 sends you to the monitor when the message "HERO" replaces the light byte on the display (we saw a "quick" 32 light byte value).

4. After case (1) above, examine memory locations 0800,0801; 0700,0701; 0ECF, 0ED0. You should observe, respectively, 05,1C; 05,1B; and 05,29. Account for that. What do you see at 0FF6,F7,F8? You should see 7E,06,00.

CONCLUSIONS

We have shown you how the designer of a robot system can go about assigning his own command codes to define his own robot language, and how to write an interpreter that executes those commands in machine language. Now that we know what the HERO software designers probably did in defining their "robot language" commands and in writing an interpreter to execute those commands (starting at F8AD in their ROM), we need no longer worry about how these commands get executed. We've seen the light. Just relax and use all their R.L. command codes wherever possible. That will be our attitude from here on.

It certainly makes programming life much easier if a command of 1,2, or 3 bytes (as HERO's commands are) can be used to replace the 50,100,200, or 300 (or whatever) bytes of machine language required to implement some particular robot action. The hard work behind those many bytes has been done for us by the interpreter writer.

In our own modest interpreter example above, the one byte 3D command gets accomplished by 9 bytes of M.L. (0628-0630). The one byte 7B command is realized with 30 bytes of M.L. in the interpreter (0631-064E). The one byte B3 command is executed with 3 machine code bytes (0625-27). And the one byte 93 command is accomplished in 6 bytes of machine language (064F-0654) in the interpreter. And this doesn't count the main part of the interpreter program (0600-0624) which is comprised of 37 bytes of M.L. So you can appreciate how much easier the robot language interpreter commands make your programming task (some of the powerful ones with HERO must surely involve several hundred bytes of M.L. in their interpreter). In

fact, about 7K of the 8K ROM on HERO is devoted to the interpreter program to execute their 37 different robot language commands from 02 to FD.

Still, you're not "off the hook" with machine language by any means. You're going to have to use it in most meaningful real world applications with HERO as the following chapters will demonstrate. Just a few examples: polling HERO's light and sound sensors and its ultrasonic "range" and "hits" buffers; polling memory locations 0000-0006 for information as to where specific motors are at any time; writing routines to specify the consequences of a detected motion interrupt; reading HERO's real time clock; controlling I/Os on HERO's experimental board (e.g. a line printer, an A to D converter, or controlling another robot by serial I/O and a modem over telephone lines - i.e. I/Os outside of HERO's system). All these, and more, are instances where you'll have to have a mastery of machine language when working with HERO. The following chapters will contain numerous applications demonstrating that point.

Chapters 5,6,7, and 8 which follow emphasize real world robot applications (industrial or other) with many examples and experiments. To be sure, a number of such applications have already been treated in the experiments of Chapters 2 and 3. The programs will, of necessity, be a mixture of R.L. commands and M.L. code. They will help to build the mastery that the first four chapters started to give you.

CHAPTER 5

LIGHT, SOUND, AND SPEECH WITH HERO. APPLICATIONS.

INTRODUCTION

We have already worked with HERO's light and sound sensors and its speech capabilities in various experiments in the previous four chapters. These attributes are becoming increasingly important with the modern robot. They are exploited in the experiments throughout this manual.

This chapter presents experiments with an applications flavor in which light and sound sensing and speech with HERO are specifically featured and explored.

The relevant interpreter robot language commands are as follows:

41(51): Enable (disable) the light detector.
42(52): Enable (disable) the sound detector.
71(72): Speak continue (wait), extended.
BF : Jump when speaking, extended.

41(51) and 42(52) do with one byte what 86 B0(90), B7 C2 E0, and 86 30(10), B7 C2 E0 did, respectively, in machine language (see Expts. 2-2,2-3,2-4 in Chapter 2). 71(72) are 3-byte commands that allow the synthesizing of a list of phonemes into intelligent sounds (speech). The second and third bytes specify the address in memory where the first phoneme in the list is to be found. The list must always end in the byte (mark) FF which the interpreter treats as a return to the address in your program just after the 71(72) 3-byte command. The list can be yours in RAM or it can be a list at some address in ROM where a canned phrase or speech is already stored. Addresses of such canned ROM speeches and phrases are given on pp. 77,78 of your Heath "Voice Dictionary" that comes with your voice synthesizer. Also in that dictionary are useful phoneme tables in which phoneme letter codes with corresponding processor code bytes, duration time, and examples illustrating the phoneme sounds in words are given. Those tables are on pp. 80-83 in the voice dictionary. BF is similar to the 71 command.

It is important to note that 41 enables light detector but disables sound detector, and vice versa for 42. However, because execution times of microprocessor instructions are so fast, you can weave in and out of 41 and 42 routines alternately to poll light and sound levels alternately with results that seem "on the fly" and instantaneous to the user ("transparent" to the user). He will be unaware of the fact that only one quantity (light or sound) is being polled at any given time. To him it is a "continuous" reading of light and then sound, and vice-versa. But please note: while 41 or 42 enable one or the other

detector, neither in itself does the actual reading (inputting) of light or sound level of intensity. This you must do by reading that level available at inport C240 (A to D converter is there) with a B6 C2 40 read instruction (there goes machine language again) just as we did in various experiments in Chapter 2 (see e.g. Expts. 2-3,2-4,3-8 among others).

All the experiments of this chapter have two goals in mind: 1) to gain a mastery in the use of HERO's light, sound detection, and speech capabilities, 2) and in the process to demonstrate and understand real world applications of robots using HERO as our prototyper.

EXPERIMENT 5-1

ROBOT POLLS FOR LIGHT. IF DETECTED IT REPEATS "GO AWAY, I SEE LIGHT" TILL LIGHT IS REMOVED. REPEATS IF LIGHT AGAIN DETECTED.

The following program does exactly what the above title asks for.

0400	3F	Change to R.L.
01	41	Enable light detector.
02	83	Change to M.L.
03	B6 C2 40	LDAA C240. Read light byte.
06	81 E0	CMPA E0. Is light level above E0? E0 is the critical level. (You can change it to any "sensitivity" level you wish).
08	22 02	BHI 02. Yes. Branch if higher to 040C
0A	20 F7	BRA F7. No. Loop back to poll at 0403.
0C	3F	Change to R.L.
0D	72 04 20	Speak, wait. Speak the phonemes at 0420 "go away" and then return to 0410.
10	72 FB 37	Speak the phonemes at address FB37 in ROM: "I see light." Return to 0413 from that routine.
13	83	Change to M.L.
14	20 ED	BRA ED back to 0403 and poll for light again.

PHONEME TABLE FOR "GO AWAY"

0420	1C	Phonemes for "go" (0420-22)
21	35	
22	37	
23	3E	Code for no sound.
24	71	Phonemes for "away" (0425-28)
25	AD	
26	46	
27	61	
28	61	
29	3E	Code for no sound.
2A	3E	
2B	3E	

2C	FF	Mark or code recognized by the synthesizer and interpreter as a "return" - in this case to 0410.

Please note that the phoneme codes at 0424 through 0428 cannot be found in your voice dictionary as such. That's because bits 7 and 6 of a phoneme code can be made (by user) to have any of four values; 0,0;0,1;1,0;1,1. Each combo determines one of four possible pitch or inflection levels appended to bits 5...0 which are the basic phoneme code for the sound to be synthesized. The phonemes listed in your voice dictionary are all valid for the lowest pitch level 00 and represent bits 5...0 with bits 7 and 6 = 0. By adding 40,80 or C0 to those phoneme codes we are introducing the higher pitch or intonation levels represented by b7,b6 = 0,1 or 1,0 or 1,1. Thus AD at 0425 is equivalent to 2D (W) but two inflection levels higher. 61 at 0427 and 0428 is really phoneme code 21 (AY like in "day") but one inflection level higher, etc. We added these varying amounts 40,80, or C0 to the phoneme codes in your dictionary to give pitch or inflection emphasis to various parts of the word "away". You ought to change codes at 0424-28 to the values given in your dictionary and compare how "away" sounds that way as against the sound produced with the codes as they are in the above program.

Execute the above program. With the light level less than E0 you hear nothing. Bring a light source up close to the LDR ("light dependent resistor") at the opening in HERO's head. You'll hear HERO repeat "Go away - I see light" for as long as the light source is there. Remove the light. HERO stops speaking. Bring it back and the words are repeated.

Make sure you understand the nature of the light polling routine at 0403 in the program, the break-out to 040C when the light level exceeds our (programmable) parameter E0 where speech is invoked, and then the return to 0403 to poll for light again after the words were said one time around. The whole scenario then repeats. Try changing the level E0 at 0407 to a lower value. Re-run and see what happens.

EXPERIMENT 5-2

ROBOT POLLS FOR SOUND. IF DETECTED IT REPEATS "I HEAR NOISE" TILL NO SOUND. REPEATS IF SOUND AGAIN DETECTED.

The program to accomplish the above follows:

0500	3F	Change to R.L.
01	42	Enable sound detector
02	83	Change to M.L.
03	B6 C2 40	LDAA C240. Read sound byte
06	81 10	CMPA 10. Is sound level above 10? We are taking 10 as the critical level. Change as you wish.
08	22 02	BHI 02. Yes, branch if higher to 050C.
0A	20 F7	BRA F7. No, loop back to poll at 0503.
0C	3F	Change to R.L.

```
 0D      72 05 20      Speak, wait. Speak the phonemes at 0520: "I hear noise",
                       then return to 0510.
 10      83            Change to M.L.
 11      20 F0         BRA F0. Branch back to 0503 and poll sound level again.

                        PHONEME TABLE FOR "I HEAR NOISE"

0520     15            Phonemes for "I" (0520-23)
  21     00
  22     09
  23     29
  24     3E            Code for no sound
  25     1B            Phonemes for "hear" (0525-28)
  26     3C
  27     09
  28     2B
  29     3E            Code for no sound
  2A     0D            Phonemes for "noise" (052A-2F).
  2B     35
  2C     23
  2D     09
  2E     21
  2F     12
  30     3E            Code for no sound.
  31     3E
  32     3E
  33     FF            Causes return to 0510.
```

Note that we have not used inflection levels above 0,0 (bits 7,6 in the phoneme byte). Execute the program. Nothing happens till sound is detected. If sound level goes above the 10 byte programmed at 0507, HERO says "I hear noise" and repeats it as long as, or every time, the sound level exceeds 10. The phrase is discontinued whenever the sound level goes below 10. Try other critical sound bytes (levels) at 0507.

EXPERIMENT 5-3

ROBOT RESPONDS ONLY WHEN BOTH LIGHT AND SOUND EXCEED CRITICAL LEVELS SIMULTANEOUSLY. STOPS RESPONSE WHENEVER EITHER OR BOTH LEVELS DROP BELOW CRITICAL.

The robot response to the case when both light and noise exceed critical levels simultaneously will be the statement "light and noise bug me - good bye" followed by a movement of HERO 20H units forward. If either or both levels drop below critical there will be no such response as the above. Whenever both levels are exceeded the response will occur again. In this experiment we have a good example of three "simultaneous" actions occurring: the polling for light, the polling for sound, and (if certain conditions are met i.e. sound and light above critical values) the speaking of phrases. We say "simultaneous". Actually those actions will occur separately in time but because of the speed of

microprocessor instruction execution (100,000 instructions per second, or more), the actions weave in and out from one to the other so fast as to appear to be occurring simultaneously.

POLLING LIGHT LEVEL

```
0600    3F          Change to R.L.
  01    41          Enable light detector.
  02    83          Change to M.L.
  03    B6 C2 40    LDAA C240.  Read light level.
  06    81 E0       CMPA E0.  Light level above E0?
  08    22 02       BHI 02.  Yes.  Branch to 060C to poll for sound.
  0A    20 F7       BRA F7.  No.  Branch back to 0603 to poll for light.
```

LIGHT LEVEL ABOVE CRITICAL, POLL SOUND LEVEL.

```
060C    3F          Change to R.L.
  0D    42          Enable sound detector
  0E    83          Change to M.L.
  0F    B6 C2 40    Read sound level.
  12    81 A0       CMPA A0.  Sound level above A0?
  14    22 02       BHI 02.  Yes.  Branch to 0618.  Speak phrases, move away.
  15    20 E8       BRA E8.  No.  Branch back to 0600 to poll for light again.
```

LIGHT AND SOUND ABOVE CRITICAL LEVELS. ROBOT SPEAKS THE PHRASES AT 0630 AND MOVES AWAY.

```
0618    3F          Change to R.L.
  19    72 06 30    Speak (wait) the phonemes at 0630 and return to 061C.
  1C    D3 10 20    Motor move, wait, relative (immediate).  Drive motor moves
                    forward 20H units at medium speed.
  1F    02          Abort drive motor.  Added as a safety factor.
  20    83          Change to M.L.
  21    20 DD       BRA DD.  Loop back to 0600 to poll light and sound again
                    for possible repeat action.
```

TABLE OF SPEECH PHONEMES FOR THE SENTENCE "LIGHT AND NOISE BUG ME, GOOD-BYE". (CONSULT YOUR VOICE DICTIONARY).

The following table of phonemes occupies memory locations 0630 through 0659 inclusive.

18,23,08,29,2A,3E; 2F,00,0D,1E,3E; 0D,35,23,09,21,12,3E; 0E,32,31,1C,3E; 0C,3C, 29,3E,3E; 1C,77,76,76,5E,3E; 0E,15,00,29,3E,3E,3E; FF.

Note that the last entry in the above table, FF, (end of speech), will cause a return to 061C where the drive motor moves forward and then aborts. Polling of light at 0600 is then resumed.

Execute the above program in the absence of light and sound. The robot remains in a state of no action. Now introduce a light source. Still no action. Remove the light source and utter sound (loud). Again no action. Now cause both light (high level) and sound (loud) to be present simultaneously. HERO

will recite the words given in the phoneme table at 0630 and then move forward and stop. Keep the light source and high level of sound still present. HERO will repeat the above action. Now remove either the light or sound source or both. HERO goes back to a quiet, inactive state. Try again. Vary the programmable critical light and sound levels at 0607 and 0613 in the program and repeat the experiment.

PROBLEM

Modify the above program to solve the problem of having HERO say "light bugs me" or "noise bothers me" or "light and noise bug me, good-bye" for the three respective cases of a) light above critical level but sound not; b) light below critical level but sound above critical; c) (as above) light and sound above critical levels. This would represent a true OR situation compared to the experiment as we formulated it which represents an AND situation.

We remind you to read Appendix B carefully for the explanation of D3 10 20, or indeed of any motor move command with the format C3 (or D3 or CC or DC) SS XX. SS is the byte that defines which of the 8 motors is to be invoked and at what speed and (in relative motion) in what direction, while XX is the byte that defines either absolute position to be reached (for the C3 or CC commands) or relative distance to be moved (for the D3 or DC commands). See also our Appendix A "Programmer's Information Sheet" reproduced from p.115 of your technical manual. Thus D3 10 20 invokes the drive motor at medium speed in the forward direction (i.e. 10=00010000. The first three 0s pick the drive motor. The 10 after that will cause motion at medium speed and the 0 after that (D bit) determines forward direction. The next two bits are involved with distance to be moved (only for the drive motor) which we choose to be 00). The 20 in D3 10 20 is the distance to be moved (not position to be moved to) i.e. D3 is a relative motor move command. Thus D3 10 20 means drive motor is to move forward at medium speed a distance of 20H units. Again, read Appendix B carefully. It is most useful in coming up with the correct SS byte (2nd byte in the command) for your needs. XX needs no explanation although Appendix B gives the limits on its absolute range for the various motors on HERO.

EXPERIMENT 5-4

THE SLEEP COMMAND 87: LOW POWER STATE OF INACTIVITY. POLLING FOR LIGHT AFTERWARDS. APPLICATION TO "GUARD DUTY".

The "Sleep Immediate" command has the format 87 XX XX. XXXX=0001 will cause a sleep period of 10 seconds, 0002 of 20 seconds, 0010 of 160 seconds etc. The sleep state is characterized by all power being shut off except to RAM memory so that your program is not destroyed. It is a most convenient feature allowing the saving of battery charge in HERO during long periods when we are not asking the robot to perform any activity. When the robot comes out of its programmable sleep period all power is restored and it can perform any programmed activity we may ask of it, then go back to sleep till the next period of activity, etc. Please note that if the robot is to obey the sleep command code 87 in your program, you must have the "sleep/normal" switch on HERO's experimental board in the "sleep" position upon program execution.

Our program below will have the robot "sleep" for 10 seconds, check light intensity when it wakes up, say "light" if detected, go back to sleep for another 10 seconds (light or not) and repeat the process indefinitely. This is one example of an application in the area of robot guard duty where long periods of time may be involved where no activity is demanded, hence battery power conservation is essential.

```
0200    3F              Change to R.L.
  01    87 00 01        Sleep for 10 seconds.
  04    41              Wakes up. Enable light detector
  05    83              Change to M.L.
  06    B6 C2 40        LDAA C240. Read light level.
  09    81 E0           CMPA E0. Light level above E0?
  0B    25 F3           BCS F3. "Branch if carry set". No, light level below E0.
                        Carry is set. Branch back to 0200 and put robot back to
                        sleep.

                              SPEAK THE WORD "LIGHT".

020D    3F              Yes, light level above E0. Change to R.L.
  0E    72 02 15        Speak phonemes at 0215 and return to 0211.
  11    83              Change to M.L.
  12    7E 02 00        JMP 0200 to put robot back to sleep. The word "light"
                        was spoken.
  15    18 23 08 29 2A 3E FF.  (phonemes).
```

Execute the program with the sleep/normal switch in the sleep position. Expose the robot to light. After 10 seconds of sleep it will awake and, if the light is still there, say "light" and then go back to sleep. Remove the light for about 16 seconds. This time the robot will sleep for 20 seconds, then awake, see light, and say "light" again, etc.

EXPERIMENT 5-5

MEASUREMENT AND DISPLAY OF THE TIME DURATION OF AN EVENT. THE PAUSE COMMAND 8F. ROBOT APPLICATIONS.

We shall imagine that the onset of an event causes light to be present, while its termination removes its presence. If we can measure the time that light is detected then we have measured the time duration of the event. An application example would be to have the robot measure the length of cartons as they pass by on the conveyor belt (and then have the robot, for example, remove or redirect those cartons with larger or smaller than critical sizes). Or the robot could measure the contact time of switches or relays in a quality control application (contact would trigger a light source, release would inhibit the light source) and reject those that did not meet the specs. Many other robot applications of this important experiment will surely come to mind, especially in the area of quality control (Q.C.).

In this experiment we shall use the Pause command 8F XX XX to help us count and display time. This command does nothing more than invoke a time delay program for us in HERO's interpreter. No power to HERO is turned off as was the case with the Sleep command 87. With XXXX=0001 you realize 1/16 sec. time delay while XXXX=0002 or 0010 gives 2/16 sec. or 1 sec. respectively. We shall simulate the onset of an event by turning on our light source and its termination by removing light. Indeed, this may be the actual case in practice rather than a simulation.

We shall display time duration in 1/8 sec. intervals. Our program lends itself to an easy one byte modification (in 8F XX XX) so that better or lesser time resolution can be obtained e.g. intervals of 1/16 or 1/4 or 1 sec. etc. With the Pause command as it is, the best we can do is 1/16 sec. To obtain better resolution than that you'd have to program your own time delay in machine language based on instructions we've used several times for that purpose: CE XX XX, 09, 26 FD, and choose XXXX as small as you wish. The following program will display the HEX number of 1/8 sec. time intervals that elapse between the onset of the event (light) and its termination (no light).

0700	C6 00	LDAB 00. ACCB initialized to 00. It will count number of elapsed time intervals when the event starts.
02	3F	Change to R.L.
03	41	Enable light detector
04	83	Change to M.L.
05	B6 C2 40	LDAA C240. Read light level.
08	81 A0	CMPA A0. Is it larger than A0?
0A	25 F9	BCS F9. No. Carry set. "Branch if carry set" back to 0705 and continue polling for light.
0C	3F	Change to R.L. Light level greater than A0. Event started.
0D	8F 00 02	Pause Immediate. 0002 will cause delay for 1/8 sec. (Change as you wish).
10	83	Change to M.L. 1/8 sec. over.
11	5C	INCB. One more time interval elapsed with light present. Increment time counter ACCB.
12	17	TBA. Move (B) to ACCA.
13	BD F6 4E	JSR REDIS.
16	BD F7 AD	JSR OUTBYT. Display number of 1/8 sec. elapsed time intervals (HEX number).
19	B6 C2 40	LDAA C240. Poll light detector.
1C	81 A0	CMPA A0. Light level still larger than A0?
1E	22 EC	BHI EC. Yes. Light still there. Branch if hi back to 070C for another 1/8 sec. pause.
20	BD F7 77	JSR INCH. No. Light gone, event over. Display shows final event duration time in HEX units of 1/8 sec. Press any key to clear display.
23	BD F6 5B	JSR CLRDIS. Blanks display.
26	20 D8	BRA D8. Branch back to 0700 to reinitialize ACCB to 00 and wait for onset of another event (light) to measure its time duration.

Note how we are using both BCS(25) and BHI(22) in polling for light. In one case (BCS), we branch back to poll again if no light detected (carry set), while in the other case (BHI) we branch back also, but this time, because light is there ("higher than"), the branch back is now for the purpose of having another 1/8 sec. elapse.

Execute the program. Nothing happens till light is detected by HERO's LDR ("light dependent resistor"). At that point the display shows the count-up of time in 1/8 second intervals till the light source is removed. At that time the display "freezes" with the time duration of the event shown as the HEX number of 1/8 sec. intervals that elapsed. Pressing any key will clear the display and ready you for the measurement of a new event's time duration. Introduce light again for a finite time and see a repeat of the scenario for the new event. Try other time "resolution bytes" at 070F.

PROBLEMS

1. Have the robot do a useful task all the while it is polling for light at 0705-0B. Use "look ahead" motor move commands CC or DC.

2. If duration time of the event reaches a critical, programmable value have the robot do something that simulates rejecting the bad contact, or the oversized carton, or what-have-you i.e. a QC application.

3. Measure lengths of cartons and sort them by size (take length = time).

EXPERIMENT 5-6

WHENEVER IT GETS DARK, ROBOT SWINGS ARM UP AND FLICKS WRIST TO TURN LIGHT SWITCH ON, THEN DROPS ARM TO NORMAL POSITION. REPEATS EVERY TIME LIGHT LEVEL RECEDES BELOW CRITICAL.

NOTE

In working with the arm/gripper/wrist on your HERO some of the experiments as written in this manual may or may not work depending on whether your HERO is "right or left-armed". It is "right-armed" if the shoulder swings the arm up from right to left; "left-armed" if the shoulder swings the arm up from left to right (looking at HERO with the keyboard in front of you). Ours was "right-armed". If yours is different, you'll have to modify the program in certain places (a good exercise).

The program follows. It is self explanatory. The procedure section after the listing discusses what you'll see. Make sure robot is initialized (Reset, 3,1).

```
0200      41          Enable light detector.
  01      83          Change to M.L.
  02      B6 C2 40    LDAA C240. Read light level.
  05      81 80       CMPA 80. Light level above 80?
  07      24 F9       BCC F9. Yes. Carry is clear. "Branch if carry clear" back
                      to 0202. Light level still above 80.

                  LIGHT LEVEL BELOW 80. HAVE ROBOT TURN LIGHT
                      SWITCH ON AND ANNOUNCE "I SEE LIGHT".

0209      3F          Change to R.L.
```

0A	C3 90 60	Motor move, wait, abs (immed). Send wrist pivot down and under to position 60H at medium speed.
0D	C3 50 65	Send arm up to 65H, medium speed.
10	C3 90 80	Flick wrist pivot up from position 60 to position 80, medium speed.
13	C3 90 00	Pivot wrist back down and under to original position 00, medium speed.
16	C3 50 00	Send arm back down to initial position 00, medium speed.
19	72 FB 37	Speak phonemes at address FB37 in ROM: "I see light." Return to 021C
1C	83	Change to M.L.
1D	7E 02 02	JMP 0202 to poll light level again for dark situation in which case switch will be turned on again.

PROCEDURE

With light on run the program in "Repeat Mode" (press Reset, keys A,D,0200). (Consult Appendix B for the C3 90 XX and C3 50 XX motor move command instructions and formats). As long as light is on (above 80 level), nothing happens. When the light level goes below 80, you'll see robot action that involves the wrist pivoting down and under, the arm going up, the wrist flicking up and down to trip a "switch", then the arm going back down. Robot announces "I see light" and then does nothing till it gets dark again, when the whole action of turning light on with the ensuing announcement repeats.

PROBLEM

Modify the program so that if the robot "misses the switch" in flicking its wrist or if the "light bulb is burned out" (in either case, the light does not go on) the robot will not say "I see light" after the arm and wrist go to their initial positions but rather something like "it's still dark - bad bulb or I'm tipsy". It then waits a definite time in its initial position, even though it's dark, and then tries again on the assumption that "someone put in a new bulb" (or it is in a more sober state). Hint: you'll have to do light polling at 0219 and, on the basis of whether light is detected or not, go on to say "I see light" etc, or go on to say "it's still dark - bad bulb or I'm tipsy" and wait, say 20 seconds, before going back up to turn the switch on again and make the light test again.

EXPERIMENT 5-7

EACH TIME SOUND LEVEL (e.g. TV SET) GOES ABOVE CRITICAL LEVEL, HERO SAYS "TURN THE TV SET DOWN".

The (A) solution to the problem is given in the following program:

0900	3F	Change to R.L.
01	42	Enable sound detector
02	83	Change to M.L.
03	B6 C2 40	LDAA C240. Read sound level.

06	81 E0	CMPA E0. Sound level greater than E0?
08	25 F9	BCS F9. No. Carry set. "Branch if carry set" back to 0903 and keep polling sound level.
0A	3F	Change to R.L. Yes, sound level above E0.
0B	72 09 11	Speak phonemes at 0911 and return to 090E.
0E	83	Change to M.L.
0F	20 EF	BRA EF. Branch back to 0900 after speaking the phonemes to poll sound level again.

TABLE OF PHONEMES FOR "TURN THE TV SET DOWN"
(CONSULT YOUR VOICE DICTIONARY).

The following table of phonemes is located in memory address space 0911 through 092B.

2A,3A,2B,0D,3E; 38,32,23,3E; 2A,2C,3E; 0F,2C,3E; 1F,02,00,2A,3E; 1E,15,23,2D, 0D,3E; FF.

Run the program. Raise the sound level above critical (E0). Hear the robot say "turn the TV down." Lower then raise the sound level. Hear the words again.

EXPERIMENT 5-8

A CONVEYOR BELT PROBLEM: FILLING CARTONS. ABSENCE OF LIGHT INDICATES CARTON TO BE FILLED IS PRESENT. PRESENCE OF LIGHT INDICATES FILLED CARTON MOVED ON AND WAITS FOR NEXT CARTON TO BE FILLED.

FORMULATION

1. When a "carton" (or can) is in place on the "conveyor belt", light is blocked. The robot goes to carton or container, opening and closing its gripper repeatedly (squeezing juice from fruits?).

2. When the carton or container is filled, a "switch is tripped" to move the conveyor and take the carton away unblocking the light which now reaches the robot's eye. We shall not here simulate the switch action - that will be done later in a very realistic and more general treatment of the problem (see Expt. 8-4).

3. In response to detecting light (carton gone), the robot turns its head around and closes its gripper. As long as no carton is present the robot maintains that inactive stance away from the conveyor belt.

4. When the robot detects the presence of another carton to be filled (light is again blocked), it goes back to the belt (i.e. turns its head back). It then again opens and closes its gripper "squeezing fruits" till the belt carries the newly filled carton or container away.

The program follows.

POLL FOR LIGHT

```
0500    41          Enable light detector
  01    83          Change to M.L.
  02    B6 C2 40    LDAA C240.  Read light detector.
  05    81 A0       CMPA A0.  Light level above A0 (carton not present)?
  07    24 0A       BCC 0A.  Carton not present, light detected (carry clear),
                    "branch if carry clear" to 0513 to turn robot away and
                    stop its activity filling the carton.
```

CARTON PRESENT (NO LIGHT). ROBOT ACTIVE FILLING THE CARTON.

```
0509    3F          Change to R.L.
  0A    C3 A8 50    Motor move, wait, abs. (immed.).  Open gripper slowly to
                    50 (see App. B).
  0D    C3 A8 30    Close gripper slowly to 30.
  10    83          Change to M.L.
  11    20 EF       BRA EF.  Branch back to 0502 to sense light and make an-
                    other decision.
```

LIGHT DETECTED, NO CARTON PRESENT, TURN ROBOT AWAY AND STOP ITS ACTIVITY.

```
0513    3F          Change to R.L.
  14    C3 C8 50    Move head CCW slowly to 50 (see App. B).
  17    C3 A8 00    Close gripper slowly.
```

POLL FOR LIGHT.

```
051A    83          Change to M.L.
  1B    B6 C2 40    Read light byte.
  1E    81 A0       CMPA A0.  Light level greater than A0 (no new carton)?
  20    24 F9       BCC F9.  Yes.  Branch if carry clear to 051B.  Poll again.
```

NEW CARTON ARRIVED. ROBOT GOES BACK INTO ACTION.

```
0522    3F          Change to R.L.
  23    C3 C8 62    Turn head slowly CW back to 62 to face "conveyor belt".
  26    83          Change to M.L.
  27    20 E0       BRA E0.  Branch back to 0509 to resume robot activity at
                    the carton on the belt.
```

Execute the program and see the gripper opening and closing repeatedly as long as there is no light. The head faces forward. As soon as light is detected (above level A0 in this case) the head turns away (CCW) and the gripper closes and remains closed for as long as the robot detects light. When light is again blocked (new carton arrived), head turns back to the forward position and the gripper commences its open/close ("squeezing") action again at the new carton till it is filled and moves away, when the events above repeat.

EXPERIMENT 5-9

A ROBOT ON GUARD - FRIEND OR FOE.

FORMULATION

1. The robot is in the sleep state (87 command).

2. It awakes every so often (here every 10 seconds) and says "go" and then moves forward and back once, creating the appearance of someone present. It keeps repeating this sequence of sleep followed by talk and motion.

3. If light is detected while moving (only the master knows where the robot eye is and what its threshhold level is) the robot instantly stops moving and says "welcome - ready".

This application will involve the powerful "branch if motor busy" interpreter command (motor here being the base drive). It allows another activity to be undertaken (in this case light polling) while the motor in question is still active. It will, therefore, of necessity be used in conjunction with the "motor move, continue" interpreter command which starts a motor and continues on to the next instruction in the program, unlike the "motor move, wait" command in which the motor runs and the next program instruction is not executed until the motor finishes the action commanded of it. The "branch if motor busy" and the "motor move, continue" commands make a very powerful team as they allow "look ahead" programming or polling for robot control, all the while the present robot action is being undertaken and implemented. We've used these commands on one or two occasions before. The program follows.

SLEEP, SPEAK, MOVE, POLL FOR LIGHT, REPEAT.

```
0600    3F          Change to R.L.
  01    87 00 01    Sleep command (10 secs.)
  04    72 06 35    Speak "go" phonemes at 0635.
  07    DC 08 10    Drive forward 10H units (slowly) but continue with next
                    instruction. See App. B.
  0A    83          Change to M.L.
  0B    8D 09       BSR 09. Branch to subrountine at 0616 to test for light.
  0D    3F          Change to R.L.
  0E    DC 0C 10    Drive back 10H units slowly but continue on with next in-
                    struction (see App.B).
  11    83          Change to M.L.
  12    8D 02       BSR 02. Branch to subroutine at 0616 to test for light.
  14    20 EA       BRA EA back to 0600. Repeat.
```

LIGHT POLLING ROUTINE WHILE DRIVE MOTOR GOES FORWARD OR BACK.

```
0616    3F          Change to R.L.
  17    41          Enable light detector.
  18    83          Change to machine code.
  19    B6 C2 40    Read light level while driving forward or back.
```

```
1C      81 E0       CMPA E0. Light level greater than E0?
1E      22 05       BHI 05. Yes. Branch if higher to 0625: "welcome - ready".
20      3F          No. Change to R.L. Light not detected.
21      1C F4       Branch if base (drive motor) busy back to 0616 to keep
                    polling.
23      83          Base reached forward or reverse limit and is no longer busy.
                    Change to M.L.
24      39          RTS. Return from sub-routine to 060D or 0614. Drive motor
                    reached its forward or reverse destination.
```

LIGHT DETECTED. ROBOT STOPS AND SAYS "WELCOME - READY"

```
0625    3F          Change to R.L.
  26    02          Abort drive motor
  27    72 06 2B    Speak "welcome" phonemes at 062B.
  2A    3A          Back to monitor. Says "ready".
```

PHONEME TABLES FOR THE WORDS "WELCOME" AND "GO".

```
062B    2D,02,00,18,19,32,23,0C,3E; FF. (Welcome).
0635    1C,35,37,3E; FF. (Go).
```

Execute the program with the sleep/normal switch in the sleep position (because an 87 command is to be executed). The robot sleeps in a low power drain state for 10 secs. It awakes, says "go", moves forward and back 10H units, goes back to sleep for 10 more seconds before awakening and repeating "go" and the forward/reverse motion. At some time while the robot is awake and moving, shine light at its LDR. It will instantly halt its motion and say the words "welcome - ready". This is a good example of a robot being used for guard duty. It welcomes its master and goes back to the executive (monitor) state only when the master (who knows the "secret" as to where the LDR is located and who has programmed for a much higher than ambient light level-here E0) shines light at its LDR. You can, of course, lengthen the sleep time.

PROBLEM

Think up a case where the robot knows that the intruder is a "foe" and not the master and have it engage in appropriate offensive or defensive actions. You can employ sound or motion or ultrasonic ranging (see Chapter 6 for the latter two) or any combination to detect the "foe". You might wish to poll range with a scanning (turning) head, etc.

EXPERIMENT 5-10

THE "MOTOR MOVE CONTINUE" (CC OR DC) VS. THE "MOTOR MOVE WAIT" (C3 OR D3) COMMANDS. "SPEAK CONTINUE" (71) VS "SPEAK WAIT" (72) COMMANDS.

We have employed and discussed some of the differences between the "motor move continue" vs. "wait" commands. (See, for example, Expt. 5-9). It is now appro-

priate to devote an experiment that goes into these commands rather thoroughly. We wish to observe their differences in action. Strictly speaking this chapter treats light, sound, and speech with HERO, but the time has come to discuss CC or DC vs. C3 or D3 and 71 vs. 72 in some detail. We take up the matter here. See Appendix A and p.115 of your technical manual for definitions of CC,DC,C3,D3,72,71. One of the observations we will make is that no two stepper motors on HERO can run simultaneously (a detailed study of your "Block/Interconnect Diagram" will indicate why). Remember, all motors associated with HERO's 7 degrees of freedom are steppers (extend/retract, arm, wrist rotate, wrist pivot, gripper, head, steer). The main drive motor for base forward/reverse motion is a DC permanent magnet type motor. It will turn out from the observations that any stepper and the main drive motor can be active simultaneously.

The CC or DC commands activate a motor and while that motor is active the program goes immediately on to the next command or instruction to be executed. However, if the two contiguous (program-wise) CC or DC commands involve stepper motors, then the second CC command does not get executed till the first one is completely executed. But if the second CC or DC command involves the drive motor then it does get executed all the while the first command is being executed so that stepper and drive motors can be active simultaneously. Thus, as we shall observe, if the first command is CC or DC and involves a stepper and the next instruction (command or Op Code) does not involve any stepper then, while the first CC or DC command is being executed, the second command or instruction will also be executed. If the first command involves the drive motor (DC) then the next instruction will be executed while the drive motor is active (thus, for example, drive and steer can work together - obviously a necessity for lateral motion).

The following program, when executed, will illustrate all the above points. What we expect to observe is discussed immediately after the program (see table Appendix B).

```
0900    CC 50 30    Move motor, continue. Send arm up to position 30.
  03    CC A8 30    Open gripper to 30, continue.
  06    72 FA 64    Speak, wait "hello, I am...robot".
  09    CC D0 82    Turn head to 82, continue.
  0C    DC 08 10    Drive forward (rel.) 10, continue.
  0F    72 FA 64    Speak, wait "hello, I am...robot".
  12    CC 50 00    Send arm down to 00, continue.
  15    CC F0 59    Steer right to 59, continue.
  18    DC 0C 10    Drive back 10 (rel.), continue.
  1B    71 FA 64    Speak, continue "hello, I am...robot".
  1E    CC A8 00    Close gripper, continue.
  21    CC 70 90    Rotate wrist to 90, continue.
  24    CC D0 62    Turn head to 62, continue.
  27    DC 08 10    Drive forward 10, continue.
  2A    CC F0 39    Steer left to 39, continue.
  2D    72 FA 64    Speak, wait.
  30    CC 90 30    Pivot wrist to 30, continue.
  33    DC 0C 10    Drive back 10, continue.
  36    CC F0 49    Steer right to 49, continue.
  39    CC 90 00    Pivot wrist to 00, continue.
  3C    72 FA 64    Speak. Wait.
  3F    CC 70 4D    Rotate wrist back to 4D, continue.
```

```
42      83          Change to M.L.
43      86 0F       LDAA 0F.
45      B7 C2 20    STAA C220.  0F to LEDs at C220.
48      3F          Change to R.L.
49      3A          Return to monitor ("ready").
```

WHAT TO EXPECT WHEN YOU EXECUTE.

1. Arm rises to 30.
2. After arm reaches 30, gripper opens to 30.
3. While gripper opens, robot speaks.
4. When finished speaking, head starts to turn to 82.
5. While head turns, HERO drives forward 10 units.
6. While HERO drives forward it speaks.
7. After speaking it sends arm down to 00.
8. When arm stops moving down, robot steers right to 59.
9. While it steers it drives back 10.
10. While it drives it speaks.
11. While it speaks, gripper closes (71 command).
12. After gripper closes, wrist starts to rotate.
13. After wrist stops rotating, head turns back to 62.
14. As head turns, robot moves forward and steers to 39 (left).
15. Robot also speaks while moving forward and steering.
16. After speaking, wrist pivots to 30.
17. While pivoting, robot drives back and steers right to 49.
18. After steer is completed, wrist pivots back to 00.
19. While pivoting, robot speaks.
20. After speaking, robot rotates wrist back to 4D.
21. While wrist is rotating, LEDs on experimental board light up and robot says "ready".

Redo the experiment with C3 (or D3) replacing CC (or DC) everywhere and again observe. All the observations should allow you to reach the same conclusions we made at the outset of the experiment. Note that driving and steering can occur together but no (program-wise) consecutive steppers can act together.

Perhaps a shorter program, easier to observe while executing, would be helpful. Try this one.

```
0800    CC 50 40    Send arm to 40, continue.
  03    CC 28 30    Extend arm to 30, continue.
  06    CC D0 82    Rotate head to 82, continue.
  09    DC 08 10    Move forward 10, continue.
  0C    CC F0 29    Steer left to 29, continue.
  0F    CC 50 00    Send arm to 00, continue.
  12    CC 28 00    Retract arm to 00, continue.
  15    83          Change to M.L.
  16    86 0F       LDAA 0F.
  18    B7 C2 20    STAA C220.  0F to LEDs at C220.
  1B    3F          Change to R.L.
  1C    3A          Back to monitor - "ready".
```

When the above is executed, you should observe the following:

1. Arm rises to 40.
2. After arm reaches 40, arm extends to 30.
3. After arm extends to 30, head rotates to 82.
4. While head rotates to 82, robot drives forward 10 and steers left to 29.
5. After head reaches 82, arm starts down to 00.
6. After arm reaches 00, it retracts to 00.
7. While arm is retracting LEDs at C220 go on and robot says "ready".
8. If CC at 0812 is changed to C3 then LEDs won't go on nor robot say "ready" till arm is fully retracted.
9. Change all CC and DC commands to C3 and D3, run and observe.

Again the observations should lead to the same conclusions we've already made. We shall come back many times to the "look-ahead" commands CC and DC in the experiments in the remaining chapters. Their use can give a robot enormous control power.

EXPERIMENT 5-11

SEARCHING FOR LIGHT. HEAD TURNS CW, CCW AND STOPS WHEN LIGHT FOUND. RESUMES SCAN WHEN LIGHT DISAPPEARS.

In this experiment with many useful applications, the head keeps turning CW and CCW through a programmable scan range. It will stop turning whenever light is found (thereby pointing to the source of light). If the light source is removed the head continues its scan from the point where it stopped. It will stop again if light is again detected and resume motion if it again disappears.

Again the "branch if motor busy" and the "motor move, continue" commands will be used as powerful allies in "look-ahead". Watch also, for our use of a semaphore to "flag" the direction the head was rotating at the time the detection of light stopped it. Such a "flag" will be necessary if we wish the head to continue to rotate in that direction if and when the light vanishes.

HEAD MOTION

0600	3F	Change to R.L.
01	CC D0 80	Turn head CW to 80, continue on.
04	83	Change to M.L.
05	C6 00	LDAB 00. Use ACCB as a flag (semaphore). B=00 means head is rotating CW.
07	8D 0B	BSR 0B to 0614 to test for light.
09	3F	Change to R.L.
0A	CC D0 44	CW head limit reached. Return from polling routine and turn head CCW to 44. Continue.
0D	83	Change to M.L.
0E	C6 FF	LDAB FF. FF in ACCB means head is now rotating CCW.
10	8D 02	BSR 02 to 0614 to poll for light.
12	20 EC	BRA EC. CCW head limit reached. Return from polling routine to here and branch back to 0600 to turn head CW again.

LIGHT DETECTION POLLING ROUTINE.

```
0614    3F          Change to R.L.
  15    41          Enable light detector.
  16    83          Change to M.L.
  17    B6 C2 40    Read light sensor.
  1A    81 E0       CMP E0.  Light above E0?
  1C    22 05       BHI 05.  If yes, branch to 0623 to abort head motion.
  1E    3F          Change to R.L.  No light detected.
  1F    1E F4       Branch if "arm" (in this case head) busy back to 0615 to
                    continue light polling while the head is still rotating.
  21    83          Change to M.L.  Head reached its CW(80) or CCW(44) limit
                    and (instantaneously) is not busy.  Fall through 1E F4.
  22    39          RTS.  Return from subroutine to either 0609 or 0612 to
                    send head back the other way and poll again.
```

LIGHT DETECTED, ABORT HEAD MOTION, SPEAK "LIGHT", POLL LIGHT DETECTOR.

```
0623    3F          Change to R.L.
  24    04          Abort arm motors (specifically head).
  25    72 06 3A    Speak "light" (phonemes at 063A).
  28    83          Change to M.L.
  29    B6 C2 40    Read light sensor.
  2C    81 E0       CMPA E0.  Light still there?
  2E    22 F9       BHI F9.  Yes.  Branch to 0629.  Keep polling with head
                    stopped.
```

LIGHT DISAPPEARED. RESUME HEAD MOTION IN SAME DIRECTION IT WAS GOING FROM POINT AT WHICH IT STOPPED.

```
0630    8E 0E D0    LDS 0ED0.  Prevents stack build-up as RTS not used here.
  33    C1 00       CMPB 00.  Is 00 in ACCB (indicating CW motion at time
                    head stopped)?
  35    27 C9       BEQ C9.  Yes, 00 found in B.  Branch back to 0600 and
                    start to turn head again CW to 80 from where it stopped.
  37    7E 06 09    No, 00 not found in B.  Must have been FF indicating head
                    motion was CCW when head stopped.  JMP back to 0609 and
                    start to turn head again CCW to 44 from where it stopped.
```

PHONEME TABLE FOR WORD "LIGHT."

```
063A    18,23,08,29,2A,3E,FF.
```

PROCEDURE

Execute the program and watch the head go through a few CW/CCW rotations (scanning). Then direct light at the robot's LDR. When the head meets the light's rays it stops rotating and says "light". It remains stopped as long as light is there. As soon as the light source is removed, the head resumes motion in the same direction it was going when stopped. It will stop again when it finds light, etc.

PROBLEM (TOUGH)

Have the robot's head track the light source as it moves to and fro around the LDR.

EXPERIMENT 5-12

A ROBOT PARTS-PLACING OPERATION IN MANUFACTURE OR CONSTRUCTION, GUIDED BY A LIGHT SOURCE.

FORMULATION

1. Robot opens gripper and waits for a switch to go high. This will alert the robot that a part is present for it to grasp. (Think of the arrival of the part as depressing a switch near the robot making its presence known). We shall use our switch S3 (bit D3) at inport C2A0 which we have placed on our experimental board.

2. When S3 goes hi, the gripper is to close, simulating the taking of a part.

3. The arm is to then swing from its down position 00 up to 80H. If light is detected on the way up, the arm is to stop, gripper to open (simulating the placing of the part at the elevation where light was found). It is then to go down to arm position 00 to fetch a new part (the presence of which is again to be indicated by S3 going hi). If no light is detected on the way up, the arm is to start back down from position 80 with the part still in its gripper. If light is detected on the way down, it will stop there to place the part by opening its gripper and then continue down to 00 to get a new part.

4. If no light is detected on the way up or down the arm returns to 00 with the part in its gripper and immediately goes back up (and down) looking for light. It will keep doing this until and where light is found, placing the part there before returning to 00.

5. Once the gripper has placed a part at the elevation ("landing" if you like) where it finds light, it is to return to 00 but does not poll for light on the way back down to 00 after placing the part. It only polls if it has a part.

6. The arm, after placing a part and returning to 00 for a new part, will stay at 00 with the gripper open until S3 is thrown high indicating presence of a new part to be taken. The arm then heads up looking for light with the gripper holding the part.

The program follows.

OPEN GRIPPER AND POLL SWITCH S3 FOR PRESENCE OF A PART.

```
0800      3F          Change to R.L.
  01      C3 A8 30    Open gripper slowly to 30H.
  04      83          Change to M.L.
```

```
05      B6 C2 A0        LDAA C2A0.  Read switches.
08      85 08           BITA 08.  Switch S3 (bit 3) high?
0A      27 F9           BEQ F9.  Switch S3 low (result of BITA 08 was 00).  No part
                        present.  Branch to 0805 to poll S3.

                    S3 HI-A PART IS PRESENT.  TAKE IT WITH
                       GRIPPER.  START SWING OF ARM UP.

080C    3F              Change to R.L.  S3 went hi.
0D      C3 A8 10        Close gripper slowly to 10 simulating taking a part.
10      CC 50 80        Motor move, continue.  Start arm up to 80 and continue on
                        to poll for light.

                        BSR TO POLL FOR LIGHT ON WAY UP.

0813    83              Change to M.L.
14      8D 09           BSR 09.  Branch to subroutine at 081F to poll for light as
                        arm swings up.

                NO LIGHT FOUND ON WAY UP.  SEND ARM BACK DOWN.

0816    3F              Change to R.L.  Arm reached 80 and no light was found.
                        Light polling routine returns to here (39 at 082D).
17      CC 50 00        Motor move, continue.  Send arm back down and immediately
                        continue to next instruction to poll for light.

                        BSR TO POLL FOR LIGHT ON WAY DOWN.

081A    83              Change to M.L.
1B      8D 02           BSR 02.  Branch to subroutine at 081F to poll for light
                        as arm swings back down from 80.

              NO LIGHT FOUND ON WAY DOWN.  SEND ARM RIGHT BACK UP.

081D    20 ED           BRA ED.  Arm reached 00.  No light found on way up or down.
                        Light polling routine (39 at 082D) returns to here.  Branch
                        back to 080C where gripper is kept in semi-closed position
                        10.  Command at 0810 sends arm right back up with the part
                        again to look for light.  All repeats.

                            THE LIGHT POLLING ROUTINE.

081F    3F              Change to R.L.
20      41              Enable light detector.
21      83              Change to M.L.
22      B6 C2 40        Poll light sensor.
25      81 E0           CMPA E0.  Light above E0?
27      22 05           BHI 05.  Yes.  Branch to 082E to place the part.
29      3F              Change to R.L.  No light yet.
2A      1E F5           Branch if arm busy back to 0821 to continue to poll for
                        light.
2C      83              Change to M.L.  Arm reached upper(80) or lower(00) limit
                        without light being found.  (Arm not busy at these ex-
                        tremes).  Fall through to 082C.
```

2D	39	RTS. Return from subroutine to either 0816 or 081D from where the BSR was made. This will tell arm to go back down or up, with the part, in search of light.

LIGHT FOUND. STOP ARM, PLACE THE PART, GO BACK DOWN (NO POLLING FOR LIGHT), GET ANOTHER PART AND REPEAT THE CYCLE.

082E	3F	Change to R.L.
2F	04	Abort arm. Stop swing. Light found.
30	C3 A8 30	Open gripper to place part.
33	C3 50 00	Motor move, wait. Send arm back down to get another part. No polling for light.
36	83	Change to M.L.
37*	8E 0E CE	LDS 0ECE. Set stack pointer at 0ECE (SP).
3A*	86 00	LDAA 00.
3C*	B7 0E D0	STAA 0ED0. Load location 0ED0 (SP+2) in stack with 00.
3F*	86 08	LDAA 08.
41*	B7 0E CF	STAA 0ECF. Load location 0ECF (SP+1) in stack with 08.
44*	39	RTS. Return from subroutine to 0800 i.e. RTS pulls 08 from 0ECF and 00 from 0ED0 and loads them into the PC, forcing a return to 0800 and never allowing the stack to grow as a 7E 08 00 would. At 0800 we repeat the whole cycle i.e. get another part and place it where light is found.

*IMPORTANT NOTE.

The routine starting at 082E is entered from the light-polling subroutine at 081F (see BHI 05 at 0827) which in turn was called by the BSR (branch to subroutine) twice in the main program (at 0814 and at 081B). We ended the light polling routine at 082D with a return (RTS-39) as we must - it being a subroutine. Since the routine at 082E is an extension of that subroutine (being entered by one of the instructions in that subroutine i.e. BHI 05 at 0827), we must also end the 082E extension of the subroutine in a RTS-39 instruction. However, our aim at the end of the 082E extension routine is to return to 0800 to get another part and place it where light is found i.e. to repeat the whole cycle. We can satisfy the requirement of ending the subroutine (here its extension) in a return (otherwise the stack will creep up if a 7E 08 00 is used and eventually, after many cycles, "wipe us out") and having the return take us to 0800 instead of to the location after where the BSR in the main program was written, by the instructions placed at 0837 through 0844. Remember that RTS pulls the byte (08) from SP+1 (0ECF) into PCH and the byte (00) from SP+2 (0ED0) into PCL, forcing a return (here) to 0800. Remember SP was set to 0ECE at 0837. This method will be used several times again in future experiments. See also Expt. 3-6.

PROCEDURE

Execute. The robot opens its gripper and waits till you throw switch S3 high. It then closes the gripper (simulates taking a part). The arm starts up. Let it go up and down a few times (00 to 80 to 00 etc.) without incident light. On the way up shine light. The arm will stop and gripper open to "place the part". Arm then goes back down to 00 and waits for S3 to go hi when it closes gripper

partially to take a new part (it will do so at once if S3 is already hi indicating a new part is available). The action repeats. Again let it do a few complete up and down swings. This time shine light on the way down. Arm stops, gripper opens to place part and arm then continues on down to 00 to get another part. Next time shine light on the first swing up or the first swing back down (if no light found on the first swing up). The series of events will be the same as those described above.

As you can imagine, this very practical experiment has many applications in manufacturing or construction.

PROBLEM

Add as many features to the program as you can think of to make it as realistic as possible. Incorporate the robot's extend/retract, wrist pivot/rotate, head rotate, etc. capabilities. Re-formulate and re-program.

CHAPTER 6

MOTION DETECTION AND ULTRASONIC RANGING WITH HERO. APPLICATIONS.

INTRODUCTION

In this chapter we shall explore HERO's motion detection and ultrasonic range measuring capabilities. Both the motion and ultrasonic range detection systems depend on the employment of sonar. In the case of motion detection, an ultrasonic wave is transmitted to the region surrounding HERO by means of an ultrasonic transmitter transducer (35kHz signal). The receiver transducer gathers in the signal reflected from nearby objects. If the reflecting object moves, a low frequency amplitude modulation is impressed on the received (reflected) signal (because of a slight variation in target range). A low pass filter in the receiver circuit allows only this modulation frequency to pass through to the receiver's final stages and this will cause a 1 to be output there. This 1 is input to line D1 of the interrupt input port C200 and indicates a "motion detect" interrupt (see e.g. Expt. 3-4).

The output of port C200 (the NANDing of all 8 possible interrupt sources to C200) then causes an interrupt into the IRQ pin of the 6808 microprocessor. As discussed in detail in Expt. 3-5, the monitor polls port C200 to pin-point the source of the interrupt and, if it finds it to be a motion interrupt, causes a jump to RAM address 0027 where the user is permitted to write his own 3-byte jump out of 0027,28,29 to a RAM location of his choice where he can write a "motion interrupt service routine" of whatever nature and for whatever purpose he decides. Further details on the circuitry and hardware associated with the ultrasonic motion detector can be found on p.77 of your Heath technical manual. Our aim here is to show how to handle the software needed to service a motion interrupt for various kinds of robot applications with HERO.

In the case of ultrasonic measurement of range to a nearby object, the ultrasonic transmitter sends out a short 32 kHz ultrasonic pulse and then measures the time till the first echo of the pulse returns from the nearest object. The distance to that object is then proportional to the reflection time measured. Refer to P.75 of your technical manual for the circuitry and hardware associated with the ultrasonic ranging transmitter. The ultrasonic receiver is, of course, also involved in the range measurement. It waits for the first echo back of the transmitter's pulse and then reads an input from the sonar timer (inport C220-see Expt. 2-9) which has been counting up from the moment the transmitter sent out the original pulse. In that way elapsed time from transmit to reception of pulse can be measured. Further details on the ultrasonic transmitter-receiver circuit-

ry and hardware can be found on p.76 of your technical manual and on p.7-44 of your Heath course notes ("Robotics and Industrial Electronics").

Our interest here will be in how to write programs that allow useful applications to be had with HERO's range measurement system. In that connection it is important to note that the range measuring circuitry and hardware together with HERO's monitor in ROM team up to constantly update addresses 0010 and 0011 in RAM. 0010 stores number of target hits detected and 0011 stores the range measured to the nearest object at any time.

The point for us to keep in mind is that we now have a handle in dealing with detection of motion around HERO, and with range measurement to objects near HERO. It consists, in the case of motion detection, of writing programs centering around the interrupt vector to 0027 in RAM; and, in the case of target range determination, of reading RAM buffer addresses 0010 and/or 0011. Examples of this will be given in the various applications experiments presented in this chapter.

EXPERIMENT 6-1

ROBOT SPEAKS AND MOVES FORWARD EACH TIME MOTION IS DETECTED.

This application could involved elements of robot safety in an industrial setting or the service that robots could perform in non-industrial applications ("guard duty"). The robot sits in a wait loop (which could represent a busy task that the robot performs) and comes out of that loop to speak "wait, something moved" and move forward only upon detection of motion. It then returns to resume its busy loop. It does this every time motion is detected.

We are now face to face with that sometimes tricky subject of interrupts. Watch carefully how we handle the motion detect interrupt which vectors us to 0027 in RAM. We'll have our program itself write the jump instruction 7E 03 50 (interrupt routine to be serviced at 0350) at vector entry address 0027,28,29. We shall also be working with the stack pointer (SP) to effect a clean return from the end of the motion subroutine at 0350 back to 0300 (the beginning of our main program) so that motion can again be detected if it occurs. We shall also be using intepreter commands 4B and 5B: "enable motion detector" and "disable motion detector".

Machine language Op Code 8E will also be used. It takes the 2 bytes (operands) appended to it and has them loaded into the stack pointer (second byte to SPH, third byte to SPL where H,L=high, low bytes), thereby setting the base address of the stack. Please remember (see Expt. 3-6 and Expt. 5-12) that if SP is set to, say 0ECE (as we do in our program below) and locations 0ECF and 0ED0 are preloaded with 03 and 00, respectively, then a 39 Op Code (RTS -"return from subroutine") will pull (pop) the contents of 0ECF and 0ED0 into the program counter forcing, in that case, a clean return to 0300, the beginning of our main program. This is a most important point: you can not simply end an interrupt routine with a JMP (7E) instruction back to, say, 0300 or else the stack will eventually build after a number of interrupts are made and kill your program.

Our program follows:

```
                MAIN PROGRAM:  INITIALIZATIONS AND SIMULATION OF "BUSY TASK".

0300      3F            Change to R.L.
  01      4B            Enable motion detector.
  02      83            Change to M.L.
  03      0E            CLI.  Clear interrupt mask (enable IRQ line).
  04      86 7E         LDAA 7E.
  06      B7 00 27      STAA 0027.  Send 7E to 0027.
  09      CE 03 50      LDX 0350.  Load 0350 into index (X) register.
  0C      DF 28         STX 28.  Send 03 to 0028 and 50 to 0029 (from X reg.).
                        Direct addressing implies page 00.
  0E      20 FE         BRA FE.  Wait loop simulates busy task.  Waits for a
                        motion detect.

     MOTION DETECT INTERRUPT SUBROUTINE.  ROBOT SAYS "WAIT, SOMETHING MOVED" AND
                MOVES FORWARD (0350 ENTERED FROM 0027,28,29: 7E 03 50)

0350      3F            Change to R.L.
  51      5B            Disable motion detector to avoid multiple interrupts.
  52      72 FB 56      Speak the phonemes at FB56 in ROM ("wait, something moved")
                        and return to 0355.
  55      D3 10 08      Motor move, wait.  Drive robot forward 08H units at medium
                        speed.
  58      83            Change to M.L.
  59      8E 0E CE      LDS 0ECE.  Load SP with 0ECE.
  5C      86 00         LDAA 00
  5E      B7 0E D0      STAA 0ED0.  00 to 0ED0 in RAM.
  61      86 03         LDAA 03
  63      B7 0E CF      STAA 0ECF.  03 to 0ECF in RAM.
  66      39            RTS.  Return from subroutine.  Pulls contents of top of
                        stack (03 and 00) into the PC for a return to 0300 in the
                        main program.
```

Execute the above program. Wave at HERO. It replies "wait, something moved" and goes forward 08H units. Wave again. The above is repeated as many times as HERO detects motion. It does nothing ("busy loop") if motion is not detected.

Study the above program carefully. There is much to be learned from it that will stand you in good stead in microcomputer and microcontroller programming for I/O control in general.

EXPERIMENT 6-2

COUNT-UP AND DISPLAY OF NUMBER OF TIMES MOTION IS DETECTED. APPLICATIONS.

In this experiment the robot keeps track of the number of those events producing motion near it. Those events could be number of people passing by, number of car-

tons passing it on a conveyor belt, the number of pieces or parts arriving for assembly, the number of finished items moving away, etc. We shall have the microcomputer store and update the number of such motion-inducing events in a memory location (0700 here), and also show each update on the display. The program follows.

INITIALIZATIONS AND BUSY TASK.

```
0500    7F 07 00    CLR 0700 in memory.  0700 will count motion detects.
  03    0E          CLI.  Enable IRQ line.
  04    86 7E       LDAA 7E.
  06    B7 00 27    STAA 0027.  Send 7E to 0027.
  09    CE 05 20    LDX 0520.  0520 to X register.
  0C    DF 28       STX 28.  05 to 0028 and 20 to 0029.
  0E    3F          Change to R.L.
  0F    4B          Enable motion detector.
  10    83          Change to M.L.
  11    20 FE       BRA FE.  Wait loop ("busy task").  Waits for motion in-
                    terrupt.
```

MOTION DETECTED. SERVICE ROUTINE TO COUNT AND DISPLAY NUMBER OF MOTION EVENTS

```
0520    B6 07 00    LDAA 0700.  Read 0700 (number of motion events) into ACCA.
  23    4C          INCA.  Increment ACCA (number of motion events).
  24    B7 07 00    STAA 0700.  Store updated count back in 0700.
  27    BD F6 4E    JSR REDIS.
  2A    BD F7 AD    JSR OUTBYT.  Display updated count.
  2D    39          RTS.  Return from subroutine (to 0511) to wait for another
                    motion interrupt.
```

Execute the program and initiate motion near the robot. Every time another motion event is detected, the display records the updated number of motion detects. Applications were discussed just before the main program. Note that the instructions at 0504-050D ensure that the motion detect interrupt vectors to 0520 via 0027.

EXPERIMENT 6-3

A ROBOT SAFETY APPLICATION: ROBOT'S ACTIVITY IS INTERRUPTED BY NEARBY MOTION. IT DRIVES AWAY AND ONLY DRIVES BACK TO RESUME ACTIVITY WHEN A KEY ON THE KEYBOARD IS PRESSED.

We shall simulate the activity in this experiment by the display of FF and a "busy loop" (at 0415 in the program below). When motion is detected, this busy loop is interrupted and activity ceases as evidenced by FF on the display changing to 00 and the robot driving forward a few units. Only when any key on the keyboard is struck (except Reset) will the robot drive back to the sight of activity and resume said activity (with FF again showing on the display). The aspect of human safety in this application should be clear.

MAIN PROGRAM: INITIALIZATIONS AND ACTIVITY.

```
0400    86 7E       LDAA 7E.  7E to ACCA.
  02    97 27       STAA 27.  7E to 0027 (direct addressing).
  04    CE 04 20    LDX 0420.  0420 to X register.
  07    DF 28       STX 28.  XH=04 to 0028; XL=20 to 0029 (direct addressing).
  09    3F          Change to R.L.
  0A    4B          Enable motion detector.
  0B    83          Change to M.L.
  0C    0E          Enable IRQ line.
  0D    86 FF       LDAA FF.  FF to ACCA.
  0F    BD F6 4E    JSR REDIS.
  12    BD F7 AD    JSR OUTBYT.  Show FF on display.
  15    20 FE       BRA FE.  "Busy" loop.
```

MOTION DETECTED. STOP ACTIVITY. MOVE ROBOT AWAY.

```
0420    3F          Change to R.L.
  21    5B          Disable motion detector.
  22    D3 10 04    Move forward 4H units.
  25    02          Abort base drive motor.
  26    83          Change to M.L.
  27    86 00       LDAA 00.
  29    BD F6 4E    JSR REDIS.
  2C    BD F7 AD    JSR OUTBYT.  Show 00.
```

PRESS ANY KEY TO DRIVE ROBOT BACK AND RESUME ACTIVITY.

```
042F    BD F7 77    JSR INCH.  Waits for press of a key.
  32    3F          Change to R.L.
  33    D3 14 04    Drive back 4H units.
  36    83          Change to M.L.
  37    8E 0E CE    LDS 0ECE.  Sets S.P. to 0ECE.
  3A    86 00       LDAA 00.
  3C    B7 0E D0    STAA 0ED0.  00 to 0ED0 on stack.
  3F    86 04       LDAA 04.
  41    B7 0E CF    STAA 0ECF.  04 to 0ECF on stack.
  44    39          RTS.  Pulls 0400 from top of stack into PC and forces re-
                    turn to 0400 to resume activity.  FF will show on display.
```

Execute the program. FF shows on the display (simulating activity). Initiate motion near the robot. The robot shows 00 on the display (cessation of activity) and moves forward 4H units. It remains that way until you press any key when it goes back 4H units, stops, and shows FF on the display indicating activity resumed. Further nearby motion causes the above series of events to repeat.

You may want to insert 02 (abort base drive motor) at 0436 and move all else down one address. This will guarantee that the robot stops after driving back 4H units.

EXPERIMENT 6-4

MOTION DETECTED (ARRIVAL OF A PIECE PART). ROBOT MOVES FORWARD, GRABS PART, RAISES AND LOWERS ARM (PLACING PART), DRIVES BACK AND AWAITS ARRIVAL OF NEW PIECE.

This is a good example of a robot application in which the robot awaits the arrival of a part (which causes a motion interrupt) and then enters some activity concerned with that part. The activity here will consist of the robot moving forward, raising and lowering its arm to place the part (you can program in the taking of the part), and moving back to await the arrival of another part (another motion detect). We leave it to you to supply mnemonics and commentary where they are omitted.

MAIN PROGRAM: INITIALIZATIONS AND WAITING FOR A PART.

```
0200    86 7E
  02    97 27
  04    CE 02 10
  07    DF 28
  09    3F
  0A    4B
  0B    83
  0C    0E
  0D    20 FE
```

MOTION DETECTED. ROBOT GOES FORWARD, RAISES AND LOWERS ARM, GOES BACK TO WAIT FOR ANOTHER PART.

```
0210    3F
  11    5B
  12    D3 10 04
  15    02
  16    C3 50 40
  19    C3 50 00
  1C    D3 14 04
  1F    02
  20    83
  21    8E 0E CE
  24    86 09*          *You could also take 00 instead of 09.  Why?
  26    B7 0E D0
  29    86 02
  2B    B7 0E CF
  2E    39              RTS.  Returns to 0209.  Why?  Motion detector re-enabled
                        at 020A to detect arrival of another part.
```

Run the program. Nothing happens till HERO detects motion (arrival of a part). It then goes forward 4H units, raises and lowers its arm (taking and placing the piece), goes back 4H units and waits for the arrival of another part (new motion detect) when it repeats the above series of events.

PROBLEM

Add any and all features to make the above application as realistic as possible.

EXPERIMENT 6-5

ROBOT MOVES FORWARD ONLY WHEN NEAREST TARGET IS GREATER THAN A CRITICAL DISTANCE AWAY, OTHERWISE STOPS. RESUMES MOTION IF TARGET RECEDES. APPLICATION OF ULTRASONIC RANGING.

Robot applications involving aspects of our program abound: 1) robot moves ahead to search for a part and stops when the sonar detects the part at a certain critical distance away from its gripper; 2) robot must incorporate safety features with respect to humans that may wander too close to the robot; 3) robot must find a clear path out of a congested area; etc.

The formulation of the problem is as stated in the title to this experiment. We shall also display the distance to the nearest target even as the target may be receding or approaching. Remember that the sonar transmit/receive boards in HERO's ultrasonic ranging system measure echo time to/from the nearest target and convert this to a relative HEX "distance" and store same in RAM memory location 0011. We therefore have the means of reading distance to nearest target at any time by reading the contents of address 0011. This we do in what follows.

ROBOT MOVES FORWARD

```
0400      3F          Change to R.L.
  01      45          Enable sonar ranging.
  02      DC 08 01    Move motor relative, continue. Send robot forward 1 unit
                      and poll target distance.
  05      83          Change to M.L.
```

READ NEAREST TARGET DISTANCE AND DISPLAY SAME.

```
0406      B6 00 11    LDAA 0011. Read (0011) for distance.
  09      BD F6 4E    JSR REDIS.
  0C      BD F7 AD    JSR OUTBYT. Show distance on display.
```

IS TARGET DISTANCE LESS THAN OR EQUAL TO A CRITICAL VALUE?

```
040F      81 30*      CMPA 30*. Distance less than or equal to critical? (* Here
                      30 is taken as critical distance. Choose any value).
  11      23 02       BLS 02. Yes. Branch if less than or same to 0415 to stop
                      drive.
  13      20 EB       BRA EB. No, branch back to 0400 and move motor forward
                      another unit.
```

TARGET DISTANCE LESS THAN OR EQUAL TO CRITICAL. STOP FORWARD DRIVE.

```
0415      3F          Change to R.L.
  16      02          Abort drive motor.
  17      83          Change to M.L.
  18      20 EC       BRA EC. Branch back to 0406 and poll target distance
                      again. It may have moved.
```

Execute the program. Use your book as a target. Keep it out of range. Robot moves. Hold the book a distance away from the target. Distance to target shows on display. Robot continues to move and will stop when the book is detected at 30H or less units away. The display will then read 30 or less. Remove the book. Robot moves forward again, with range greater than 30 showing on the display. Bring the book into range again. Robot again moves forward till it comes within 30H of the target. You can observe how accurate this ultrasonic ranging system actually is by withdrawing your book to a distance that shows a read-out of 31 or 32 on the display. The robot will move and then stop with the display showing 2D or 2F or 30. Withdraw the book very slightly till the display again shows 31 or 32. The robot will immediately move forward till the display value decreases to 2D or 2F or 30 and then stop, etc. ("cat and mouse").

EXPERIMENT 6-6

ROBOT STOPS ONLY IF TARGET DISTANCE IS LESS THAN CRITICAL OR LIGHT IS SENSED.

This is a two-sensor problem: an OR situation. Many robot applications should come to mind.

DRIVE ROBOT FORWARD

```
0500    3F          Change to R.L.
  01    45          Enable sonar.
  02    41          Enable light detector.
  03    DC 08 01    Move motor, relative, continue.  Drive robot 1 unit forward.
  06    83          Change to M.L.
```

POLL NEAREST TARGET DISTANCE AND LIGHT SENSOR. DISPLAY TARGET DISTANCE.

```
0507    B6 00 11    Read target distance.
  0A    BD F6 4E    JSR REDIS.
  0D    BD F7 AD    JSR OUTBYT.  Display distance.
  10    81 30*      CMPA 30*.  Distance less than 30?
  12    23 09       BLS 09.  Yes.  Branch to 051D to abort drive.
  14    B6 C2 40    LDAA C240.  No.  Poll light sensor.
  17    81 80       CMPA 80.  Light level greater than 80?
  19    22 02       BHI 02.  Yes.  Branch to 0510 to abort drive.
  1B    20 E3       BRA E3.  No.  Neither target range less than critical nor
                    light found.  Branch to 0500 and keep driving robot forward.
```

DISTANCE TO TARGET LESS THAN CRITICAL OR LIGHT FOUND. ABORT DRIVE.

```
051D    3F          Change to R.L.
  1E    02          Abort drive motor.
  1F    83          Change to R.L.
```

20	7E 05 07	JMP 0507 to poll for target distance and light level all over. Distance to target and light condition may have changed. If so robot may have to move forward again.

Execute the program. With no target present and no light, robot moves forward. If light is detected, robot stops even if target distance is greater than critical. If light is removed, robot drives forward till target distance is less than critical(*) then stops. Withdraw target. Robot again moves till light is found OR target distance is again less than critical.

Remember that critical distance and light level parameters can be changed at 0511 and 0518 in the above program.

PROBLEM

Modify the program so that the robot stops only if target distance is less than critical AND light is detected.

EXPERIMENT 6-7

ROBOT SCANS FOR FIRST TARGET OUTSIDE A CRITICAL RADIUS ("EARLY WARNING SYSTEM").

When the rotating head finds the first target whose distance is greater than critical, it stops scanning and points in that direction, with the distance to target showing on the display (elements of an "early warning system" - approach of an intruder (safety)).

In the program that follows we again use the powerful duo of commands: CC and 1E. CC allows a motor (here the head rotate) to go through the distance indicated in the command and at the same time continue with the next instruction. If the next instructions involve polling (in our case reading target range with the ultrasonic system) then that polling will continue if the 1E (branch if arm (here head) busy) command is used in that polling until (in our case) the head finishes its rotate. Then something else can be done - whatever we please (in our case, we rotate the head back in the other direction if no target at a distance greater than critical was found). The program follows.

ROTATE HEAD TO SCAN FOR TARGET (BACK AND FORTH).

0600	3F	Change to R.L.
01	45	Enable sonar
02	CC D0 82	Rotate head to 82(CW). Continue on.
05	83	Change to M.L.
06	8D 09	BSR 09. Branch to subroutine at 0611 to poll target distance.
08	3F	Change to M.L. Return to here from subroutine.
09	CC D0 42	Rotate head back from 82 to 42. 82 was reached, no target found. Continue on.

```
  0C      83          Change to M.L.
  0D      8D 02       BSR 02 to 0611 to poll target distance.
  0F      20 EF       BRA EF. Return to here from subroutine at 0611. 42 was
                      reached and no target found. Branch back to 0600 to rotate
                      head back to 82 again.

            SUBROUTINE: POLLING AND DISPLAY OF TARGET DISTANCES.

0611      B6 00 11    LDAA 0011. Read distance to nearest target.
  14      BD F6 4E    JSR REDIS.
  17      BD F7 AD    JSR OUTBYT. Display distance.
  1A      81 80       CMPA 80. Target distance greater than 80 (critical dist-
                      ance)? (Change 80 as you please).
  1C      22 05       BHI 05. Yes. Branch to 0623 to stop head rotate and
                      point to that target.
  1E      3F          Change to M.L. No. Continue head scan.
  1F      1E F0       Branch if busy back to 0611 and continue to poll for tar-
                      gets as head continues to rotate.
  21      83          Change to M.L. Head reached scan limit (either 82 or 42)
                      and, being "not busy", you "fall through" 1E F0.
  22      39          RTS. Return from subroutine to either 0608 or 060F to
                      turn head the other way.

      ABORT HEAD MOTION. TARGET FOUND AT DISTANCE GREATER THAN CRITICAL.
                    POINT TO IT AND DISPLAY ITS DISTANCE.

0623      3F          Change to R.L.
  24      04          Abort arm motor (head scan).
  25      83          Change to M.L.
  26      20 FE       BRA FE. Halt.
```

Execute the program and observe the head come to a stop and point to a target that is at a distance greater than critical (here 80) - if found. It will point along a line of sight in which targets no closer than 80 are present i.e. through an alley or hole in the "defense". You can adjust the 80 at 061B to any value that works best in your surroundings.

PROBLEMS

Modify the program starting at 0626 so that you can press any key to resume a new scan at any time (from the point at which the head rotate stopped). Since 0623 is part of the subroutine at 0611 you'll need to end this modification with RTS (39) which may mean you'll have to set the stack and have the 39 pull off the desired return address from it (we've done this before in previous experiments). You may also need to employ a "semaphore" to know in which direction the head was rotating at the time it stopped so as to pick up in the same scan direction when you press any key. Formulate and solve other target range problems like this one with other goals and aims of your own.

EXPERIMENT 6-8

ROBOT WALKS ALONG AN ENCLOSURE LOOKING FOR AN OPENING. TURNS AND WALKS THROUGH IT IF AND WHEN FOUND. ("DON'T FENCE ME IN").

This application has serious and light aspects to it. Think of some of the situations where it could be useful (industrial environments) or amusing (in the home or fenced-in lawn).

Simulate a fence by holding a book in your hand and going along with the moving robot, blocking its sonar transmit-receive channels. This simulates a fence very close by. When you remove the book, simulating an opening in the fence, or end of the fence, the robot will turn and go a certain distance "through the opening", then say "stop" and halt its motion(or HERO goes along wall and turns into a room).

ROBOT TURNS HEAD SIDEWAYS TO LOOK FOR OPENING
IN FENCE AND WALKS PARALLEL TO IT.

```
0700      3F            Change to R.L.
  01      45            Enable sonar.
  02      C3 D0 82      Turn head CW to 82 for ultrasonic beam orientation.
  05      DC 08 01      Robot drives forward 1 unit parallel to fence (slowly).
                        Continue on to next instruction.
```

ROBOT LOOKS FOR OPENING IN FENCE.

```
0708      83            Change to M.L.
  09      B6 00 11      Read range to nearest target (fence).
  0C      BD F6 4E      JSR REDIS.
  0F      BD F7 AD      JSR OUTBYT. Display distance to fence.
  12      81 30         CMPA 30. Is distance less than 30H units? i.e. fence still
                        next to robot?
  14      22 02         BHI 02. No. Fence ended or opening in it was found i.e.
                        distance greater than 30. Branch to 0718.
  16      20 E8         BRA E8. Yes. Fence still there. Branch back to 0700 to
                        drive robot forward another 1 unit along fence.
```

FENCE ENDED OR OPENING WAS FOUND. DRIVE
ROBOT AROUND FENCE OR THROUGH OPENING.

```
0718      3F            Change to R.L.
  19      C3 F0 59      Steer wheel to 59 (49 is its straight ahead position) to go
                        around fence or through the opening found.
  1C      D3 08 30      Drive robot around fence (or through opening) 30 units.
  1F      02            Abort drive motor (safety precaution).
  20      72 07 26      Speak phonemes at 0726 (the word "stop").
  23      83            Change to M.L.
  24      20 FE         BRA FE. Halt.
  26      1F,2A,15,23,25,3E,FF.  (Phonemes for "stop"). Return to 0723.
```

Execute the program. When the head reaches position 82, place your book right next to the transmit-receive channels in the head. The robot moves "along the fence", tracking it and displaying its proximity. Remove the book after the robot has moved two or three feet. This simulates the end of the fence or an opening found in it. The robot will steer around and move 30H units (going around fence or through "opening" in it). It will then halt as it says "stop".

PROBLEM

Think up and solve other variations of this experiment. At 0726 in the program have the robot say "don't fence me in".

The motion detect and ranging features studied in this chapter are powerful and useful robot attributes.

CHAPTER 7

HERO's REMOTE MANUAL CONTROL TEACHING PENDANT. THEORY BEHIND ITS OPERATION. APPLICATIONS. MODIFYING TAUGHT PROGRAMS.

INTRODUCTION

The teaching pendant is a very important and valuable accessory - one that most modern robots are provided with. On HERO it is available through input port C280 (see Expt. 2-13). It allows two useful operations that a robot should be capable of: manual control (as a tele-operator) and, through manual control, teaching and learning capability.

In manual remote control we are able manually, (as opposed to software), to control every stepper (arm, wrist, head, gripper, etc.) and put it through its paces and, as we do, even have an instantaneous read-out on the display of any stepper's position. This is most useful in trouble-shooting motor problems should they arise, or should we think they exist. Remote manual control can easily and quickly put each and every robot degree of freedom through its paces simply by setting the pendant at the appropriate switch positions calling for the corresponding motion of that motor. In that way we save valuable time and effort compared to writing the software to test the various motors and degrees of freedom. (Actually the pendant settings, as we shall see, are calling the monitor's software ("utility" programs) into action to serve us - software that drives the various motors of our choice). In that way hardware trouble-spots can be quickly pin-pointed for further remedial action. Remote manual control is also most useful where the robot must work in a hazardous environment and we, remotely, can control its actions with the pendant. (In that case it is not really acting as a robot but as a "tele-operator". Strictly speaking (RAI definition) the robot should be driven by a program (software)).

The remote control teaching pendant also allows the robot to learn and then execute the steps that were learned. The world of robots is becoming increasingly sophisticated, powerful, and intelligent as the microprocessor becomes its controlling and dominant force. In the teach mode, the pendant can put the robot through a series of desired actions or motions, as discussed above; but in this "learn" mode those actions will also be recorded in memory (thanks to the microprocessor) as equivalent software instructions and commands - the very ones you have been using and working with in this manual - in the form of a program which can reproduce those very actions when executed. Once the desired, taught robot actions are recorded in memory, not only can those actions be reproduced at will by running that taught program, but the taught program can be edited or modified so as to have the robot improve upon or extend those actions in, say,

a manufacturing or industrial environment. This lends great flexibility to the robot to cope with new requirements or situations or conditions. This powerful flexibility of the modern "smart" robot can only be had where there is a microcomputer whose memory stores the actions induced by the remote control pendant in the form of instructions, whose CPU (microprocessor) executes those instructions in the playback mode, and whose keyboard can edit, change, or re-arrange those learned instructions to give more power and versatility to the robot. Our experiments (and theory) in this chapter will illustrate all the above.

We shall not dwell on the details of all the individual settings with the teach pendant that produce individual, specific actions. These can be had by reading pp. 21-24 in the "User's Guide" that comes with your HERO, as well as pp. 58,59, 73 in your "Technical Manual". You can also find details on pendant settings in Expt. 18 of unit 12 in the Heath course notes "Robotics and Industrial Electronics" if you have them. We will present those details as needed in the course of our experiments. The aim will be to understand what's behind the "magic" of the pendant (a program, of course) and to provide practical applications experiments with the pendant.

We divide this chapter into parts (A) and (B). (A) will deal with the pendant in the remote manual (tele-operator) mode, and (B) on using the pendant in the teach mode. As always, understanding (fundamentals) as well as applications will be stressed.

PART (A): THE PENDANT IN REMOTE MANUAL MODE (TELE-OPERATOR)

If we wish to use the pendant in remote manual mode and have the monitor move appropriate motors for us through desired steps and directions corresponding to the pendant settings, we can do so with HERO by resetting the microcomputer and striking key 4 (with the teach pendant connected to the lower connector on the back panel of HERO). Pressing key 4 tells the polling monitor (reset puts it into a keyboard polling mode) that we wish to have it jump to the utility routine that will respond to the settings of the pendant to have the selected motor move in the selected direction. Let's be specific: press Reset, then key 4. See "4' on the display (initially you should set the rotary switch to neutral ("N") on the pendant). We shall discuss the cases of two specific pendant settings and the attendant actions they produce, and then account for those actions by analyzing what the software in the monitor must be doing.

CASE 1: SET ROTARY SWITCH TO "ARM-PIVOT" AND FUNCTION SWITCH TO "ARM".

You'll immediately see 00 on the display with 4 still showing. This signifies that the arm is in position 00 (down). (We assume you've initialized HERO upon reset by pressing keys 3,1 thereby invoking the initialization utility routine in the monitor). Now throw the motion switch to the right and pull the pendant's trigger. The arm starts rising and the display reads the changing arm position as it rises: 00,01,02,... When you release the trigger, arm motion and display count-up stop. Throw the motion switch left, pull the trigger and see the arm start down and the position on the display count down.

CASE 2: SET ROTARY SWITCH TO "HEAD" AND
FUNCTION SWITCH TO "ARM".

You'll now see 62 on the display alongside the 4. 62 is, of course, the head position in its normal, initialized forward straight-ahead stance. Throw the motion switch to the right and pull the trigger. You'll see the head rotate CW and the head's position, shown on the display, steadily increasing until you release the trigger. Then the head motion and count-up stop. If you throw the motion switch left and pull the trigger again, the head will rotate CCW and the head position will decrease on the display.

We shall discuss and treat cases (1) and (2) above in the very important experiment that follows. The purpose of that experiment is to gain an understanding of what the software in the monitor behind "utility mode 4" is doing to make the robot respond in the way it does to the above pendant settings. We shall gain that understanding by writing our own program that will respond to the same pendant settings as in cases (1) and (2) above and produce the same observed motions and position read-outs. If we succeed in that undertaking then we can be sure the software behind "utility mode 4" must be essentially the same as the program we propose in the following experiment.

EXPERIMENT 7-1

A PROGRAM THAT IMPLEMENTS REMOTE MANUAL CONTROL ACTIONS OF MOTORS CORRESPONDING TO THOSE REQUIRED BY VARIOUS PENDANT SETTINGS.

We are interested here in creating our own machine language program that will produce the same motor actions as does the monitor's "utility mode 4" when various settings on the pendant are input to the microcomputer through the pendant connector on HERO. Those actions should be the same as, for example, those observed in cases (1) and (2) above for the specific pendant settings we treated there. Before we can do that, we'll need to know the input codes of those pendant settings (input through HERO's inport C280). It is only through those codes that the microcomputer can identify a particular pendant setting and be able to respond with the motor motion appropriate to that setting that the user is asking for (through utility mode "4", for example). See also Expt. 2-13.

Recalling that the pendant codes are input through HERO's inport C280, the following program will allow us to read and display the various codes corresponding to various pendant settings.

```
0300    BD F6 4E    JSR REDIS.
  03    B6 C2 80    LDAA C280.  Read pendant code.
  06    BD F7 AD    LSR OUTBYT.  Display the code.
  09    CE 10 00    LDX 1000 for short time delay.
  0C    09          DEX
  0D    26 FD       BNE FD till X=0000.
  0F    20 EF       BRA EF to 0300 and poll for new pendant settings (if any).
```

When you execute this program and read the codes corresponding to the pendant settings in cases (1) and (2) above, you'll find the following results.

CASE 1: D7 (rotary switch to "arm-pivot", function switch to "arm", motion switch "right", trigger pulled. Action to be implemented: arm to move up).

CASE 2: F7 (rotary switch to "head", function switch to "arm", motion switch "right", trigger pulled. Action to be implemented: head to rotate CW).

Using the knowledge that under "utility mode 4" pendant codes D7 and F7 caused the arm to swing up and the head to rotate CW, respectively, and also caused arm or head positions to be displayed, we should be able to "dream up" our own program that will do just that in response to the pendant settings corresponding to cases (1) and (2) above. This experiment and its program should teach you a lot of what's behind the "mystery" of remote pendant control operation under the direction of software. You might come up with a better program of your own. The one presented below is powerful and yet short and simple. Read it and the commentary carefully.

```
        POLL FOR EITHER D7 OR F7 PENDANT CODES (ARM OR HEAD ACTIVITY REQUESTED).

0400    B6 C2 80    LDAA C280. Read port C280 for pendant code.
  03    81 D7       CMPA D7. Was it D7 (arm to go up)?
  05    27 06       BEQ 06. Yes. Branch to 040D
  07    81 F7       CMPA F7. No. Was it F7 (head to move CW)?
  09    27 13       BEQ 13. Yes. Branch to 041E.
  0B    20 F3       BRA F3. Neither. Branch to 0400. Keep polling pendant.
                    No motors move yet.

      PENDANT CODE D7 WAS READ. MOVE ARM UP AND DISPLAY ITS POSITION.
            ARM STOPS IF PENDANT SETTING CHANGES. CONTINUES UP
                  IF PENDANT SETTING IS AGAIN D7.

040D    3F          Change to R.L.
  0E    DC 50 01    Arm motor move, continue. Moves up 1 unit (relative).
  11    83          Change to M.L.
  12    B6 00 01    LDAA 0001. Read memory location 0001 for arm position.
  15    BD F6 4E    JSR REDIS.
  18    BD F7 AD    JSR OUTBYT. Display arm position.
  1B    7E 04 00    JMP 0400 and poll pendant code again. It may have chang-
                    ed. If so, arm will stop rising, e.g. release trigger.

        PENDANT CODE F7 WAS READ. ROTATE HEAD CW AND DISPLAY ITS
        POSITION. HEAD MOTION STOPS IF PENDANT SETTING CHANGES.
              CONTINUES CW IF PENDANT SETTING IS AGAIN F7.

041E    3F          Change to R.L.
  1F    DC D0 01    Head motor move, continue. Moves CW 1 unit (relative).
  22    83          Change to M.L.
  23    B6 00 05    LDAA 0005. Read (0005) for head position.
  26    BD F6 4E    JSR REDIS.
  29    BD F7 AD    JSR OUTBYT. Display head position.
  2C    7E 04 00    JMP 0400 and poll pendant code again. If it has changed,
                    head stops rotating (e.g. trigger was released).
```

PROCEDURE

Run the program. The robot remains inactive. Set pendant for case (1) (rotary to "arm-pivot", function to "arm", motion to "right", trigger pulled) i.e. a D7 code. Arm rises in incremental units of 1 and the display shows the incrementing arm position. Release the trigger or motion switch or both. The code has changed from D7. 81 D7, 27 06 at 0403-06 are violated (you do not go to the routine 040D in that case). Arm stops rising and position on display stops incrementing. Re-impose the setting above that leads to the code D7. The arm continues to rise from where it stopped and the position on the display increments anew. Again release trigger or motion switch and everything stops.

Now set the pendant for case (2) (rotary to "head", function to "arm", motion switch to "right", trigger pulled) i.e. a F7 code. Head rotates CW in incremental units of 1 and the display shows the incrementing head position. Release the trigger or motion switch or both. Pendant code changes from F7. 81 F7, 27 13 at 0407-0A are violated (you do not go to the routine at 041E in that case). Head stops rotating and position on display stops incrementing. Re-impose the above setting that leads to the code F7. The head will continue to rotate from where it stopped and the position on the display again increments. Release trigger or motion switch and all stops. Set the pendant for the D7 code again. The arm will rise again from its last stopped state as will arm position displayed. Try other pendant codes. There'll be no response because our program does not poll for other settings (codes). Continue playing with the pendant in and out of "Head CW" and "Arm Up" settings (F7 and D7 codes).

PROBLEM

Find all codes for all the pendant settings and write a new program that will allow any motor to move in any direction when the pendant is given the appropriate setting. That motor is to stop when trigger is released. Position of any motor, as it moves, is to be shown on the display. At any time you can change the pendant setting to stop the last motor that was moving and then invoke a new one with the new pendant setting.

CONCLUSION TO EXPERIMENT 7-1

And that is the way that manual control of the robot is accomplished. The monitor does for us what we did above ourselves when you press key 4 coming out of Reset with HERO. That puts us squarely into the "utility routine" in the monitor which will react to the various pendant settings to produce the appropriate corresponding actions for arm, head, wrist etc. It will accomplish that with a program much as ours above, in which the pendant setting is polled as to its code and, when it is read into the accumulator, is compared with all possible pendant codes. The code that matches then forces a jump to the software routine that produces the desired action. By writing such a "utility" ourselves, as we did here, we benefit from the understanding gained. Once done, we can henceforth leave it up to the monitor to identify pendant settings and produce the appropriate robot action. The robot software designer certainly had to go through all that himself in order to make the robot easier for the user to control ("user friendly") in remote operation by simply pressing key 4 after Reset.

We now move on to the application of the remote control pendant as a teaching tool in which the pendant puts the robot through known and desired paces that we wish

it to repeat in, say, some manufacturing application. It is to reproduce those actions on demand or to reproduce keyboard modified (edited) versions of those actions should the application dictate such modifications.

PART (B): THE PENDANT USED IN THE TEACH OR LEARN MODE. STORING, EXECUTING, AND MODIFYING THE TAUGHT PROGRAM.

In the learn (or teach) mode, the very actions that you induced by specific pendant settings in the manual mode of part (A) are now stored in memory automatically so that the robot will remember what it has learned. But, we ask, what is it that will be stored in memory? Surely it should be instructions and/or R.L. commands that, if they were written as a program by user, would result in the same actions that the pendant in manual mode induced with its various settings. If that indeed happens, then we can execute that learned program and have the robot reproduce modified actions should that be desirable where requirements and conditions might demand it ("modified" here means changing it from KBRD).

In this part we shall present applications experiments demonstrating the use of the pendant in its teach or learn mode. In these experiments we shall use the monitor's "utility mode 7" (press Reset, then key 7) to do the "hard work" (through one of its monitor routines) of learning the pendant induced actions and translating them into a program of software stored in memory for future recall. It must translate every action induced by the pendant into a group of corresponding machine language/robot language instructions/commands. After presenting several applications using "utility mode 7" to do this translation of robot actions into a program and storing same in memory, we shall then ask ourselves the question: "how does 'utility mode 7' do the 'magic' that it does?" We shall answer that by trying to write our own program which will, in a general way, do what "utility mode 7" does for us. In so doing, we will have learned all the intracacies of a teaching pendant and its use with robots.

Before we go into the experiments, it is necessary to outline the procedure the user has to follow when using "utility mode 7" in the teach/learn mode:

1. Press Reset and key 7. The display will show "7" and four dashes.
2. Enter the initial address at which you want the stored (translated) program to start. The display will again show 7 and four dashes.
3. Enter the last address of the stored program (make it larger than you think it will be but not so large as to encroach on any other of your programs). You'll then see 7.F.NNNN where NNNN is the initial address (typed in (2) above) plus 8. That means the translated program of learned robot actions will not be stored at the initial address in (2) above, but rather at 8 locations higher. What then gets stored at those first 8 locations? The answer is: FD 00 00 4D 00 00 62 49. "Utility mode 7" did this for us so that when we replay the stored program from the initial address typed in (2) above, the first 8 bytes will result in initializing the robot's 7 steppers to positions 00,00,4D,00,00,62,49 (extend, arm, rotate, pivot, gripper, head, steer) before a replay of their learned actions - certainly desirable. Remember FD is the R.L. command "motors, move all, absolute (immediate)".
4. You are now ready to incude any sequence of robot actions with the pendant in its various possible settings. Those actions will be stored as an equivalent

program starting at the address 8 locations higher than the initial address typed in (2) above (thanks to the "utility 7" program in the monitor). Further, after the last action requested by the pendant, the last command automatically entered in memory by "utility mode 7" is 3A (return to monitor and say "ready"). This command terminates the stored program. Of course, later, should we wish to modify the learned program we can always write in other instructions or commands in lieu of 3A.

EXPERIMENT 7-2

A LEARNED, STORED ACTION. ITS REPLAY. MODIFICATIONS FROM THE KEYBOARD.

We shall have the pendant teach the robot the following actions: 1. drive ahead, 2. move head around CCW, 3. open gripper, 4. close gripper, 5. move head back CW, 6. drive back, 7. open gripper, 8. close gripper.

The above actions might represent a typical robot manufacturing operation: go to pick up a part, come back with it, place it.

Enter "utility mode 7" starting at address 0100 with final address 0200. The display will show 7.F.0108 after final address is typed in. Use the pendant settings that correspond to each of the above actions to cause those actions to occur sequentially. After step 1 you'll see 7.F.010B on the display indicating that action 2 will be stored starting there - action 1 was stored as a 3 byte command from 0108 to 010A. After step 2, the display will show address 010E, after step 3 it will show 0111, after step 4 0114, after 5 0117, then 011A, then 011D, and finally after step 8 it will show 0120. In each case 7.F. also shows. Now reset and examine your stored program starting at 0100. We found the following (compare with Appendix B's table of motor command bytes):

```
0100     FD 00 00 4D 00 00 62 49 (move all).
  08     C3 08 04       Drive ahead 4 units slow.
  0B     D3 CC 02       Turn head CCW 2 units (rel.) slow.
  0E     D3 A8 15       Open gripper 15 units (rel.) slow.
  11     D3 AC 13       Close gripper 13 units (rel.) slow.
  14     D3 C8 03       Turn head CW 3 units (rel.) slow.
  17     C3 0C 04       Drive back 4 units slow.
  1A     D3 A8 10       Open gripper 10 units slow.
  1D     D3 AC 10       Close gripper 10 units slow.
  20     3A             Return to monitor ("ready").
```

Certainly these stored 3 byte command codes are in keeping with each action that we had the pendant teach the robot (see Appendix B).

Now replay the program you found (first perform a "32" initialization i.e. Reset then keys 3,2). You'll see the robot replay the actions exactly as they occurred when taught with the pendant.

MODIFICATIONS OF THE ABOVE PROGRAM.

1. At 0120 replace 3A by 7E 01 00 (JMP 0100). Execute this modified program and see the robot do the learned actions over and over again.

2. Replace the above 7E 01 00 jump by 8F 00 40 at 0120 (pause for 4 seconds) and follow it with 7E 01 00 at 0123. Run this modified program and see the robot do one cycle (steps 1-8), then pause for 4 seconds, then redo the cycle, pause, redo, etc. indefinitely.

3. We'll now modify the learned program so that the robot repeats the above cycle of actions a finite number of times (interspersed with time delay) and outputs to the display the count-up of the cycles as they unfold. To accomplish that we write a "patch" at 0140 as follows:

```
0140      C6 00          LDAB 00.  ACCB counts cycles.
  42      7E 01 00       JMP 0100
```

Then at 0120 write the following add-on to the learned program:

```
0120      83             Change to R.L.
  21      5C             INCB.  Increment cycle counter.
  22      17             TBA.  Transfer B to A.
  23      BD F6 4E       JSR REDIS.
  26      BD F7 AD       JSR OUTBYT.  Display number of cycles completed.
  29      81 04          CMPA 04.  Have 4 been done?
  2B      27 08          BEQ 08.  Yes.  Branch to 0135 and stop.
  2D      3F             Change to R.L.  No, 4 not done.
  2E      8F 00 20       Pause for 2 seconds.
  31      83             Change to M.L.
  32      7E 01 00       JMP 0100 to do another cycle.
  35      3F             Change to R.L.
  36      3A             Return to exeutive.  Says "ready".  Cycles stop.
```

Execute this modified program from 0140 (A,D,0140). You'll see the robot do each cycle, display the number of cycles as they are executed, pause for 2 seconds between cycles, do another cycle, update the cycle count on the display, pause again, etc. The byte at 012A determines the number of cycles to be executed (we chose 4 here). You can make it anything you wish.

4. You can, of course, modify the stored program so that any or all positions traveled to, or distances covered, by the motors (e.g. bytes 04,02,15, etc. at addresses 010A,010D,0110, etc.) can be changed. The same type of action will be performed but over different distances - again a most useful flexibility feature to have with robots to cope with new demands.

PROBLEM

Modify the program in (3) above, starting at address 012D, so that you can abort the robot action after the present cycle is completed by throwing switch S3 on the experimental board (S3 must be high when the present cycle is completed). Try to work it out for yourself before looking at our answer below.

ANSWER

At 012D in (3) above change the instructions to: B6 C2 A0, 85 08, 26 08 and then continue with 3F, 8F 00 20,83,7E 01 00,3F,3A as before. Be sure to change 08 at 012C to 0F. Analyze these additions and changes and make sure you understand them. Ability to abort robot action is a most important feature to have.

This experiment shows you some of the real power of the teaching pendant and how your modifications of the resulting stored program can add enormous versatility and flexibility to what the robot has learned. This makes it easily adjustable to new or changing conditions or requirements that the application may demand. Again an example of the power that the microcomputer provides in creating "smart robots".

EXPERIMENT 7-3

ANOTHER TEACH PENDANT APPLICATION EXAMPLE.

Consider the following robot actions to be learned for a real-world industrial environment type of application.

1.Drive ahead (medium speed), 2. move head CCW (medium speed). 3. open gripper (fast), 4. extend arm (fast), 5. close gripper partially (fast) e.g. around an object, 6. retract arm with object, 7. move head back CW (medium speed), 8. drive back (medium speed), 9. extend arm (fast), 10. open gripper (fast) releasing object, 11. retract arm (slowly), 12. close gripper completely (slowly).

We shall leave it to you to implement all the pendant settings required to realize these individual, sequential actions. Use "utility mode 7" as was done in Expt. 7-2. After you do, reset and examine the program that the utility mode stored in memory in response to the various manual pendant settings and motor actions. Compare every one of the 3-byte commands that now comprise the stored program with what you expect, based on the actions you invoked. See Appendix B for help in this regard.

After you have examined the stored program, run it. The robot should do what it was taught, once, and then return to the monitor and announce "ready".

Now write modifications around the stored program which will have the robot do the following:

1. Repeat the action (cycles) indefinitely.
2. Repeat them indefinitely but interspersed with time delay between cycles.
3. Repeat the cycles a finite, programmable number of times, still with time delay between cycles.
4. Abort the cycles when a switch (on your experimental board) is thrown high (after the present one is finished).
5. Modify distance or position bytes to have the robot do the same action but modified as to positions.

EXPERIMENT 7-4

WHAT'S BEHIND THE TEACH PENDANT IN LEARNING AND MEMORIZING A PROCESS. WRITING YOUR OWN UTILITY PROGRAM TO DO WHAT "UTILITY MODE 7" IN HERO's MONITOR DOES.

As our final task in this chapter we ask ourselves: "What's going on anyway? How did 'utility mode 7' do all those wonderful things for us in the teach/learning mode? Can we write our own 'utility' program that will do, more or less, the same general things?" Only by writing such a program ourselves will we be able to delve into the real happenings behind the pendant's teach/learn process and thereby make this a meaningful and useful chapter.

To make life a little simpler for ourselves in this task and still get the full meaning and significance of what we are doing, we shall have our program below poll only for a "move head CW" pendant setting i.e. for the associated pendant code F7 at inport C280 (see Expt. 7-1). This setting requires you to place the rotary switch to "head", function switch to "arm", motion switch to "right" and the trigger must be pulled (see Expt. 7-1).

After you see how our utility program works and how it will succeed in programming and memorizing the action being imposed by the remote manual pendant setting, you should modify and embellish that utility program by including other codes corresponding to other pendant settings for other actions. Just follow our method here. You should also first review the program and approach behind Expt. 7-1.

In our program below we shall store C3 and D0 at memory locations 0700,01 corresponding to: C3 = motor move, wait, absolute; and D0 = head rotate CW at medium speed. We shall also store 3A at 0702. These three locations form a look-up table for the bytes that will need to be memorized and stored (at 0500, say) to duplicate the pendant induced head action. Here goes.

POLL PENDANT SETTING

```
0600      B6 C2 80      Read port C280 for pendant code.
  03      81 F7         CMPA F7.  Is it F7 (rotate head CW)?
  05      27 02         BEQ 02.  Yes.  Branch to 0609.
  07      20 F7         BRA F7.  No.  Branch back to 0600 and continue polling
                        C280.  No motor action.
```

PENDANT CODE WAS F7. STORE BYTES C3 AND D0 AT 0500,01. ROTATE HEAD CW. READ HEAD POSITION AT 0005, STORE IT AT 0502. STORE BYTE 3A AT 0503. JUMP BACK TO POLL PENDANT SETTING (CONTINUE HEAD ROTATION OR STOP IT).

```
0609      B6 07 00      Read C3 from 0700 into ACCA.
  0C      B7 05 00      Store it at 0500.
  0F      B6 07 01      Read D0 from 0701 into ACCA.
  12      B7 05 01      Store it at 0501.
  15      3F            Change to R.L.
  16      DC D0 01      Rotate head CW, rel., 1 unit.
  19      83            Change to M.L.
  1A      B6 00 05      Read head position.
```

```
1D      B7 05 02      Store it at 0502.
20      B6 07 02      Read 3A from 0702 into ACCA.
23      B7 05 03      Store it at 0503.
26      7E 06 00      JMP 0600 and poll pendant code again. This will reveal
                      whether continued head motion is desired or whether it
                      should stop (e.g. was trigger or motion switch released?)
```

BYTE TABLE

0700-02: C3,D0,3A.

PROCEDURE

1. Run the program from 0600 (Reset, 1,D,0600). Nothing happens.
2. Set the pendant for a "head rotate CW" operation i.e. code F7.
3. See the head turn CW.
4. Release the trigger and/or motion switch. Head stops rotating.
5. Reset. Examine locations 0500-0503. See C3,D0,XX,3A where XX is the last value of the head position (read from 0005) when the head motion stopped. Since we are rotating CW, XX could be anything from 62 to BF depending on how long you allowed the head to rotate.
6. Initialize the robot with the "32" utility mode thereby sending the head back to its straight-ahead position 62.
7. Run the memorized, stored program from 0500 (execute by keying in A,D, 0500 after Reset). See the head duplicate "to a T" the motion we had the pendant teach it. When it stops (at the position we taught it to stop at) the robot goes into the monitor and says "ready". Our "utility" program at 0600 allowed all the pendant controlled action to be memorized and stored starting at 0500 for future replay or modification.

A MODIFICATION OF OUR UTILITY PROGRAM

We can modify the above program so that as the head turns under pendant control, the head position (updated) shows on the display. To do this write the following changes after head position is read at 061A:

```
061D      BD F6 4E      JSR REDIS.
  20      BD F7 AD      JSR OUTBYT.  Display head position.
  23      B7 05 02
  26      B6 07 02
  29      B7 05 03
  2C      7E 06 00
```

We can also modify the taught program stored at 0500 so that when it is replayed the final head position will show on the display. This can be done as follows.

```
0500      C3 D0 XX      This is the memorized program corresponding to the pendant -
                        induced action. XX=final head position. Add the following
                        to it.
  03      83            Change to M.L.
  04      B6 00 05      Read final head position.
  07      BD F6 4E      JSR REDIS.
```

```
0A      BD F7 AD      JSR OUTBYT.  Display it.
0D      20 FE         BRA FE.  Halt.
```

PROBLEM

Extend our "utility program" to include the possibility of polling other pendant settings for other motor actions to be produced by the pendant and stored for future replay.

CONCLUSION

And there you have it! With Experiments 7-1 and 7-4 you have learned in rather careful detail just how a teaching pendant operates and does the things it does. From here on you are free to use "utility modes 4 and 7", relying on the monitor to do those things for you by means of its own programs equivalent to those given in Experiments 7-1 and 7-4 here.

CHAPTER 8

APPLICATIONS OF AN INDUSTRIAL OR MANUFACTURING NATURE (SOME FUN AND GAMES TOO).

While many of the experiments to this point have, in themselves, applicability to industrial or manufacturing operations, those presented in this chapter are wider in scope, more advanced, and lend themselves to greater flexibility. They have an unmistakable manufacturing or industrial thrust. Experiments 8-9, -10, and -11 also have the element of "fun and games" about them.

Several HERO robot language commands not used hitherto will be introduced and be responsible for a leap forward in the power, flexibility, and real robot world applicability of the experiments. These are the "motor move, wait (continue) abs. indexed", E3(EC), commands and the "motor move, wait (continue) abs. extended", F3(FC), commands.

We believe the experiments in this chapter will also demonstrate the value of HERO not only as a learning vehicle but as a robot that lends itself extremely well as a prototyper and experimenter. Ideas and applications can be prototyped, tried out, modified, adjusted, perfected and then carried over, with appropriate changes, to other systems.

EXPERIMENT 8-1

A GENERALIZED PROGRAM TO IMPLEMENT TEACHING A ROBOT ACTIONS FROM THE KEYBOARD. REPLAY THE ACTION ONCE OR A FINITE NUMBER OF TIMES. MODIFYING (EDITING) THE ACTION. ABORT OPTION. THE E3(EC) INDEXED COMMAND.

In this experiment we will teach the robot the course of action to be taken by inputting (while the program is running) motor select/speed and position (SS, XX) command bytes from the keyboard (MDI: Manual Data Input). This is in contrast to the teach pendant approach employed and studied in Chapter 7. The command bytes SS,XX input from the keyboard are then made, by the program, to form a table in (RAM) memory. The robot action can then be made to unfold by our use in the program of the powerful HERO motor command E3 00 or EC 00 to access that table. You are to think of the driving program as having been written for you by the "vendor" in ROM, and the user-inputted command bytes, entered from the keyboard while that program is running, as forming the SS,XX table in RAM (tables at run time, like any data acquisition bytes, can only be stored in RAM).

(in the form 01,02,...or 0F) and the cycle of actions unfold. After each cycle is done, the robot re-initializes itself and the number of cycles left to be done is shown on the display before the next cycle commences.

6. When all the requested cycles are done, "3C" again shows on the display and the operator again has the choice of enetering D,1 for "do once", or E,B for "enter new table bytes", or D,C for "do continuously" just as in (5) above.

7. An abort of the action is to be possible. As each motor goes through its action as part of one cycle (in the "DC" - "do continuous" - case), the CPU is to poll a switch (S3 on the experimental board-input address C2A0). If, during a particular motor's action, the CPU finds S3 high, "AB", standing for "Abort", replaces the number of cycles on the display, that motor finishes its action, and the robot is then re-initialized. No further cycles are done. "3C" again shows on the display. All action stops. The operator again has 3 choices: D,1 ("do once"); E,B ("enter new table"); or D,C ("do continuously"). All then proceeds as before.

The program that implements all of the above follows. Study it and the commentary very carefully. Note the use of the index register and indexed addressing with E3 and EC to access the table of motor action bytes (SS/XX) that you input from the keyboard. Keyboard input of desired robot action is an alternative to the teach pendant. It has its own advantages and conveniences. It is a powerful tool used frequently in commanding a robot for the course(s) of action desired.

```
0100      CE 01 E0      LDX 01E0.  01E0 will be base address of SS/XX byte table.

                            DISPLAY "EB" ("ENTER BYTES").

0103      86 EB         LDAA EB.
  05      BD F6 4E      JSR REDIS.
  08      BD F7 AD      JSR OUTBYT.  Show "EB".
  0B      BD F6 4E      JSR REDIS.

                  ENTER MOTOR MOVE BYTES SS,XX.  CREATE TABLE.

010E      BD F7 96      JSR IHB.  Input HEX byte SS or XX (2 keys).  Byte shows on
                        display and is in ACCA.
  11      A7 00         STAA 00 (indexed).  Store (ACCA) at memory address (X)+00.
  13      08            INX for next position in table.
  14      81 FF         CMPA FF.  Mark FF entered?
  16      27 02         BEQ 02.  Yes.  All SS,XX bytes entered.  Branch to 011A
                        for choice of action.
  18      20 F1         BRA F1.  No.  Continue entering motor bytes at 010B.

        ALL SS,XX BYTES ENTERED.  DISPLAY "2C": PROMPT FOR TWO CHOICES.

011A      86 2C         LDAA 2C.
  1C      BD F6 4E      JSR REDIS.
  1F      BD F7 AD      JSR OUTBYT.  Display "2C" (two choices).

              ENTER CHOICE "D,1" OR "E,B".  DISPLAY "D1" OR "EB".

0122      BD F6 4E      JSR REDIS.
```

```
 25     BD F7 96      JSR IHB.  Enter "D,1" or "E,B".  "D1" or "EB" shows on
                      display and are in ACCA.  "D1: do once"; "EB: enter bytes".

                 WAS IT D1 ("DO ONCE") OR EB ("ENTER BYTES")?

0128    81 D1         CMPA D1.  Was it D1?
 2A     27 09         BEQ 09.  Yes.  Branch to 0135.
 2C     81 EB         CMPA EB.  No.  Was it EB?
 2E     27 D0         BEQ D0.  Yes.  Branch to 0100 to enter new bytes for new
                      table and see "EB" on the display.
 30     BD F6 4E      JSR REDIS.  Neither D1 or EB entered.
 33     20 E5         BRA E5.  Branch to 011A.  Continue to see "2C" on display.
                      Attempt D,1 or E,B again.

                      IT WAS D1.  DO ROBOT ACTIONS ONCE.

0135    CE 01 E0      LDX 01E0.  Set X to base of table.
 38     A6 00         LDAA X (indexed).  Fetch byte from (X)+00 in the SS,XX byte
                      table.
 3A     81 FF         CMPA FF.  FF reached (robot action done once)?
 3C     27 08         BEQ 08.  Yes.  Branch to 0146.
 3E     3F            Change to R.L.  No, FF not reached.  Do next motor action.
 3F     E3 00         Move motor wait, abs, indexed.  Fetch the SS,XX bytes from
                      table and do the action.
 41     83            Change to M.L.
 42     08            INX
 43     08            INX to next SS byte in the table for next motor's action.
 44     20 F2         BRA F2.  Branch to 0138 to do next motor's action.

          ONE CYCLE OF ROBOT ACTION DONE.  SHOW "FD" ("FINISHED") ON
                          DISPLAY.  REINITIALIZE ROBOT.

0146    86 FD         LDAA FD.
 48     BD F6 4E      JSR REDIS.
 4B     BD F7 AD      JSR OUTBYT.  Display "FD" (finished).
 4E     3F            Change to R.L.
 4F     FD 00 00 4D 00 00 62 49.  Reinitialize motors.

             DISPLAY "3C".  PROMPT FOR THREE CHOICES.  ENTER CHOICE
                    "D,1" OR "E,B" OR "D,C".  DISPLAY SAME.

0157    83            Change to M.L.
 58     86 3C         LDAA 3C
 5A     BD F6 4E      JSR REDIS.
 5D     BD F7 AD      JSR OUTBYTE.  Display "3C" (three choices).
 60     BD F6 4E      JSR REDIS.
 63     BD F7 96      JSR IHB. Enter "D,1" or "E,B" or "D,C" ("do continuously").
                      Display D1 or EB or DC.  D1,EB, or DC in ACCA.

                DID YOU ENTER D1 ("DO ONCE"), EB ("ENTER
                   BYTES") OR DC ("DO CONTINUOUSLY")?

0166    81 D1         CMPA D1.  D1 entered?
```

```
  68      27 CB          BEQ CB.  Yes.  Branch to 0135 to do robot actions once again
                         (and see "D1").
  6A      81 EB          CMPA EB.  No.  EB entered?
  6C      27 92          BEQ 92.  Yes, branch to 0100 to enter new bytes (create new
                         table) and see "EB".
  6E      81 DC          CMPA DC.  No.  DC entered?
  70      27 02          BEQ 02.  Yes.  Branch to 0174 to do cycles continuously.
  72      20 E4          BRA E4.  None of the above.  Branch to 0158 to see "3C".
                         Three choices again.

              ENTER NUMBER OF CYCLES DESIRED.  DO CYCLES CONTIN-
                   UOUSLY A FINITE NUMBER OF TIMES OR ABORT.

0174      BD F7 77       JSR INCH.  Input number cycles desired.
  77      16             TAB.  Save (ACCA) in ACCB.
  78      BD F6 4E       JSR REDIS.
  7B      BD F7 AD       JSR OUTBYT.  Display number cycles wanted.
  7E      CE 01 E0       LDX 01E0.  Reinit X to table base.
  81      A6 00          LDAA 00 (indexed).  Fetch M(X+00) into ACCA.
  83      81 FF          CMPA FF.  FF reached (another cycle done)?
  85      27 0B          BEQ 0B.  Yes.  Branch to 0192 to reinit robot and do an-
                         other cycle.
  87      3F             Change to R.L.  No, do next motor move in the same cycle.
  88      EC 00          Motor move, continue, abs. indexed.  Get motor bytes SS
                         from X+00 and XX from X+01 and execute them, continue.
  8A      83             Change to M.L.
  8B      7E 01 B1.      JMP 01B1 to poll S3 for possible abort request.
  8E      08             INX.
  8F      08             INX.  No abort requested.  Prepare for next SS,XX entries
                         in table for next motor's action.
  90      20 EF          BRA EF to 0181 to do next motor move in the cycle.
  92      3F             Change to R.L.  Another cycle done, reinit all motors.
  93      FD 00 00 4D 00 00 62 49.  Move all motors.  Reinit.
  9B      83             Change to M.L.
  9C      5A             DECB.  B counts cycles left to go.
  9D      17             TBA.  (ACCB) to ACCA.
  9E      BD F6 4E       JSR REDIS.
  A1      BD F7 AD       JSR OUTBYT.  Display cycles left.
  A4      81 00          CMPA 00.  00 reached (all cycles done)?
  A6      26 D6          BNE D6.  No.  Branch to 017E, more cycles to go, do another
                         one.
  A8      7E 01 58       JMP 0158.  Yes.  All cycles done.  Display "3C" and enter a
                         choice again (D1, EB, or DC).

                   POLL SWITCH S3 FOR POSSIBLE ABORT REQUEST.

01B0      83             Change to R.L.
  B1      B6 C2 A0       LDAA C2A0.  Read switches at C2A0.
  B4      85 08          BITA.  S3=1?
  B6      26 07          BNE 07.  S3=1.  Branch to 01BF to abort.
  B8      3F             Change to R.L.  S3=0.  No abort requested.
  B9      1E F5          Branch if arm busy (any stepper except steer) back to
                         01B0 to poll S3 again while motor is still moving (busy).
  BB      83             Change to M.L.  Motor move done (not busy).
```

```
  BC      7E 01 8E        JMP 018E.  Motor move done, no abort requested.  Go to the
                          next motor move at 018E.

                                SERVICE AN ABORT REQUEST.

01BF      86 AB           LDAA AB.
  C1      BD F6 4E        JSR REDIS.
  C4      BD F7 AD        JSR OUTBYT.  Display "AB" ("abort").
  C7      3F              Change to M.L.
  C8      FD 00 00 4D 00 00 62 49.  Move all motors.  Reinit.
  D0      83              Change to M.L.
  D1      7E 01 58        JMP 0158.  Display "3C".  You have 3 choices again:  D1,EB,
                          or DC.
```

PROCEDURE

In entering the motor bytes after the program is executed (two at a time: motor select/speed byte (SS) first and position byte (XX) next), the following table will be helpful (it is based on medium speed activity of each motor-see also Appendix B):

MOTOR SELECTED	SELECT/SPEED BYTE (SS)	POSITION: INITIAL,RANGE (XX)
Drive	10	
Extend/retract	30	00,00-98
Arm (shoulder)	50	00,00-86
Wrist rotate	70	4D,00-93
Wrist pivot	90	00,00-A5
Gripper	B0	00,00-75
Head	D0	62,00-BF
Steer	F0	49,00-93

1. Run. See "EB".
2. Enter bytes two at a time (motor select/speed byte (SS) and position byte (XX)). See them displayed as entered.
3. Terminate with FF (entered from keyboard).
4. See "2C" on display (two choices).
5. Enter D,1 or E,B. See "D1" or "EB". They are requests to "do action once" or "enter motor bytes (table) again". If not D1 or EB, continue to see "2C" and try again.
6. If D,1 see "D1" and do action once (one cycle replays).
7. After one cycle is completed see "FD" on display ("finished"). Robot then re-initializes all motors.
8. When all motors are re-initialized, see "3C" on the display. You have three choices:
9. Enter either D,1; E,B; or D,C. If none of these are entered keep seeing "3C" and try again. If one of these, see "D1", "EB" or "DC" meaning either "do once", "enter new bytes(table)", or "do continuously". If D1 or EB then all goes as in (1)-(8) above.
10. If DC, see "DC" on display and key in the number of cycles desired (1,2,...F). See that number on the display. The cycles commence. After each cycle the robot re-initializes itself and then displays the number of cycles remaining.

When all the cycles are done see "3C" on the display again, i.e. three choices D1,EB, or DC available to you again, as before.

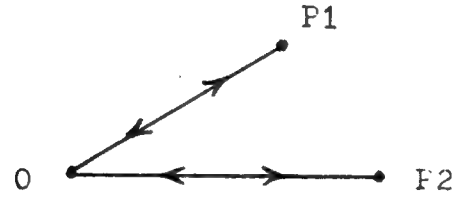
11. If you wish to abort before all the cycles are done, throw S3 high during any motor action occurring as part of one cycle. See "AB" (for "abort") on the display. The motor presently engaged finishes its action. The robot then re-initializes itself and when it does, "3C" again shows on the display. You can enter your command D1, EB, or DC again and all will proceed as before. SAVE THE PROGRAM For Expt. 8-2.

EXPERIMENT 8-2

A PICK AND PLACE ROBOT TAUGHT FROM THE KEYBOARD (BASED ON THE GENERAL PROGRAM OF EXPT. 8-1).

This experiment has wide applicability in industrial manufacturing operations. We can easily handle it as a special case of the very general program given, discussed, tested, and applied in Expt. 8-1 just concluded.

FORMULATION



1. The robot picks a part at 0, places it at P1, returns to 0, picks another part there, places it at P2, returns to 0 for another part, places it at P1, and keeps repeating.

2. As in Expt. 8-1, the cycle 0 to P1 to 0 to P2 to 0 can be done once or a finite number of times. It can also be aborted in the latter case. The action can also be modified by entering new table bytes (mode "EB": "enter bytes") just as in Expt. 8-1. You'll appreciate the generality of the program developed in Expt. 8-1.

3. By entering additional SS and XX bytes over and above those indicated in the experiment below, you can add further positions P3,P4, etc. in the pick and place operation. Note: the picking does not have to always originate at the same origin 0. It can originate at a position 01, then, when desired, at another position 02 (not shown), etc. Thus a very general "pick and place" robot operation can be easily implemented on the basis of the general program of Expt. 8-1. We shall leave it to you to generalize the procedure below by entering sufficient bytes to include additional pick points 01,02,...and place points P3,P4,...

PROCEDURE

1. Execute the program given in Expt. 8-1 at 0100. See "EB" on the display (prompt to "enter motor action bytes").

2. Enter the following bytes (in pairs SS,XX) for our pick and place action (see the motor SS,XX table in the procedure section of Expt. 8-1, or Appendix B, for medium speed):
   a) Pick action at 0: 30,30 (extend arm to 30); B0,30 (open gripper to 30); B0,00 (close gripper); 30,00 (retract arm).
   b) Go to P1: D0,52 (turn head CCW to 52 which is position P1).
   c) Place operation at P1: 30,30 (extend arm to 30); B0,30 (open gripper to 30); B0,00 (close gripper); 30,00 (retract arm).
   d) Go back to 0: D0,62 (turn head CW to 62).
   e) Pick action at 0: 30,30; B0,30; B0,00; 30,00 (same action as in (a) above).
   f) Go to P2: D0,72 (turn head CW to 72 which is position P2).
   g) Place operation at P2: 30,30; B0,30; B0,00; 30,00 (same as in (c) above).
   h) Go back to 0: D0,62 (turn head back to 62).

3. The inputs in (2) above will define one complete pick and place cycle: 0 to P1 to 0 to P2 to 0. You should now:

4. Enter FF (terminate or mark).

5. Now see "2C" on the display. You have 2 choices: either enter D,1 ("do once") or E,B ("enter bytes" to modify table).

6. If D,1 see "D1" on display. Robot goes through one complete pick and place cycle. (What happened to the motor re-init. command at 014F and 0193 in Expt. 8-1)?

7. Then see "3C" on the display. You have 3 choices: either enter D,1 ("do once"), or E,B ("enter bytes" to modify table), or D,C ("do continuously" i.e. a finite number of times).

8. If D,C see "DC" on display and key in the number of pick and place cycles (0 to P1 to 0 to P2 to 0) you desire i.e. enter key 1,2,...or F and see 01,02, ...or 0F on the display. The cycles now commence. See the count-down of the cycles remaining on the display. When all the cycles are done, see "3C" on the display. You have 3 choices again: D,1; E,B; or D,C as before (do once, modify action table, or do continuously). See (6) above for question.

9. Should you wish to abort before all the cycles are done (in the "DC" mode choice), throw abort switch S3 high and see "AB" ("abort") on the display. The robot will reinitialize itself to its "nest" (position 0) after completing its present motor move action in the cycle. You'll then again see "3C" prompting a new choice D1, EB, or DC as before.

PROBLEM

Modify the above procedure to allow for a more general pick and place robot whereby you enter SS,XX bytes in such a way that additional and distinct pick points 01,02,... and place points P3,P4,... are admitted into the robot's work plan.

NOTE

In the procedure as presented above, if E,B ("EB") is entered (steps 5 or 7), the program, of course, goes back to the beginning and you must enter all the motor select/position (SS/XX) bytes all over again even if it's just one byte you may want to change. In that case it would be simpler to do so by resetting and examining the memory locations in the table for the one or two bytes you may wish to change (remember the SS/XX table is constructed starting at 01E0-see Expt. 8-1). Thus suppose, for example, you wish to change "place position" P1 from head position 52 to head position 32. If you examine steps 2 (a),(b) in the procedure above, you'll find that 52 is at address 01E9 in the table that was constructed as you entered the SS,XX bytes from the keyboard while the generalized program of Expt. 8-1 was running. Hence a Reset followed by keys 1,E, 01E9 gets you there. You'll see 52 on the display. Press key C for "change", then 32 then F for "forward". 32 has replaced 52 at 01E9. All you now have to do is to run the program from 011A (see Expt. 8-1). "2C" will show on the display. Enter D1 ("do once") and see one cycle. Then see "3C". Enter D1,EB, or DC and all proceeds as before.

CONCLUSION

The flexibility offered by the program and approach of Expts. 8-1, 8-2 in controlling, teaching, changing, or aborting robot action from the keyboard is enormous. The rest of this chapter will explore still other approaches to added flexibility and versatility of the robot in still other important industrial type applications.

EXPERIMENT 8-3

CHOOSING FROM A MENU OF ROBOT TASKS STORED AS TABLES. THE F3 (FC) COMMAND.

In this application we enter the SS,XX (motor select and position) bytes appropriate to a given desired robot task as a table in RAM before we run any program. In fact we enter other SS,XX bytes as another table defining another task, and still other SS,XX bytes as still another table defining still another task etc., i.e. a menu of tables in RAM defining different tasks. The main program that accesses these tables to perform any one of the desired tasks from the menu is to be considered as written in ROM (even though, of course, we write it in RAM). Thus the user can set up his own menu of tasks in RAM and the main program driving and implementing the table is to be considered as having previously been written by the "vendor" in ROM. HERO's robot language command codes F3 or FC will play a very important role in this "menu" technique. In Expts. 8-1 and 8-2 the tables were entered dynamically at "run time", here they're entered pre-run time.

The F3 (or FC) command's format is F3 (or FC) MMMM. F3 (or FC) means "motor move, wait (continue) abs. extended". The necessary two bytes SS and XX determining motor select/speed and position are to be found in memory addresses MMMM and MMMM+1 respectively. Thus if, at pre-run time, you construct a table

of SS and XX bytes in pairs at definite locations in RAM, the F3 (or FC) command will fetch those two bytes at MMMM and MMMM+1 and execute them if MMMM is specified in the F3 (or FC) MMMM format. For example, if D0(SS) is at 0380 and 42(XX) is at 0381 then F3 0380 will fetch D0 from 0380 and 42 from 0381 and cause the head to rotate to 42 at medium speed. Then the command F3 (or FC) MMMM+2 will fetch the next two SS,XX motor bytes in the table to execute the next motor's desired action, etc. In this case the tables (menu) of different robot tasks or applications could easily be changed by the user by simply Resetting and entering the new tables in RAM (flexibility and versatility). The main program driving the table is to be considered as having been written in ROM. You could also construct the table dynamically at run-time (as was done in Expts. 8-1, 8-2 using the index register as a pointer), or you could have new sensed bytes input into the tables during robot execution to replace the previous tables. In that case the robot "learns from its environment" as the latter changes (see Expt. 8-8). We do not follow the dynamic approach here as we did in Expts. 8-1, 8-2, and again in 8-8.

FORMULATION

1. Three different robot tasks will be stored as 3 different tables in memory constituting our "menu". Table 1 starts at 0380 and defines the action: turn head, lift arm, lower arm, turn head back. Table 2 starts at 0390 and defines the action: pivot wrist, rotate wrist, rotate back, pivot back. Table 3 starts at 03A0 and defines the action: move forward, move back. We are defining simple actions for purposes of illustrating the method involved.

2. When the program is executed, the display will show the prompt "CC" ("choose number of cycles"). Operator then enters the desired number of cycles wanted for the task to be chosen 1,2,...or F. 01,02...0F then shows on the display.

3. The display will now also show "CA" (a prompt for "choose the action"). Operator enters 80,90,or A0 corresponding to one of the 3 tasks desired from the menu, corresponding to tables at 0380,0390, and 03A0 as described above. 80,90 or A0 will show on the display. If the operator errs and enters keys other than 80,90, or A0 he will know about it because the display will again show "CC" and he must re-enter the number of cycles and task number 80,90, or A0.

4. The task requested starts to unfold and the cycles remaining to be executed for the particular robot task requested shows on the display (updated after execution of another cycle).

5. When the number of cycles requested have been accomplished, the display again shows "CC". The operator is again requested to enter the new number of cycles desired and, in response to the prompt "CA" that appears, the next robot task 80,90, or A0 from the menu is also to be entered.

6. We are not incorporating the abort feature here. We leave it to you to modify the program below so as to have that desirable possibility (see Expts. 8-1,8-2).

The program follows (consider main driver to be in "ROM" and the task tables (menu) at 0380,0390,03A0 to have been entered by the user in RAM).

INITIALIZATIONS, PROMPTS "CC" AND "CA", CHOICE OF NUMBER OF CYCLES AND OF THE TASK DESIRED.

```
02FD    BD F6 5B    Clear display (JSR CLRDIS).
0300    86 CC       LDAA CC.
  02    BD F6 4E    JSR REDIS.
  05    BD F7 AD    JSR OUTBYT.  See CC on display.
  08    BD F7 77    JSR INCH.  Enter cycles desired.
  0B    16          TAB.  Save # of cycles desired in ACCB.
  0C    BD F6 4E    JSR REDIS.
  0F    BD F7 AD    JSR OUTBYT.  Show # cycles desired on display
  12    86 CA       LDAA CA.
  14    BD F7 AD    JSR OUTBYT.  See "CA" next to cycles.
  17    BD F7 96    JSR IHB.  Enter 80,90, or A0 (choice of task from the menu).
                    See 80,90, or A0 on the display.
  1A    81 80       CMPA 80.  Was task at 0380 table desired?
  1C    27 0A       BEQ 0A.  Yes.  Implement it at 0328.
  1E    81 90       CMPA 90.  No.  Task at 0390 table desired?
  20    27 23       BEQ 23.  Yes.  Implement it at 0345.
  22    81 A0       CMPA A0.  No.  Task at 03A0 table desired?
  24    27 3C       BEQ 3C.  Yes.  Implement it at 0362.
  26    20 D5       BRA D5.  Neither 80,90,A0 entered.  Try again at 02FD.  All
                    the above repeats.
```

TASK IN TABLE AT 0380 DESIRED (80 ENTERED). IMPLEMENT IT.

```
0328    3F          Change to R.L.
  29    F3 03 80    Move motor, wait, abs., extended.  Fetch motor bytes SS,
                    XX from 0380, 0381 in table.
  2C    F3 03 82    Fetch motor byes SS,XX from 0382, 0383.
  2F    F3 03 84    Fetch motor bytes SS,XX from 0384, 0385.
  32    F3 03 86    Fetch motor bytes SS,XX from 0386, 0387.
  35    83          Change to M.L.
  36    5A          DECB.  Another cycle done.  Decrement ACCB.
  37    17          TBA.  Retrieve cycles left from ACCB to A.
  38    BD F6 4E    JSR REDIS.
  3B    BD F7 AD    JSR OUTBYT.  Show cycles left.
  3E    81 00       CMPA 00.  00 reached (all cycles done)?
  40    26 E6       BNE E6.  No.  Do another cycle at 0328.
  42    7E 02 FD    JMP 02FD.  All cycles done.  Jump to 02FD.  See "CC".
                    Enter new number of cycles for new robot task.
```

TASK IN TABLE AT 0390 DESIRED (90 ENTERED). IMPLEMENT IT.

```
0345    3F          Supply mnemonics, commands, and commentary for this routine.
  46    F3 03 90
  49    F3 03 92
  4C    F3 03 94
  4F    F3 03 96
  52    83
  53    5A
  54    17
  55    BD F6 4E
  58    BD F7 AD
```

```
  5B      81 00
  5D      26 E6          BNE E6 to 0345.
  5F      7E 02 FD
```

TASK IN TABLE AT 03A0 DESIRED (A0 ENTERED). IMPLEMENT IT.

```
0362      3F             Supply mnemonics, commands, and commentary for this routine.
  63      F3 03 A0
  66      F3 03 A2
  69      83
  6A      5A
  6B      17
  6C      BD F6 4E
  6F      BD F7 AD
  72      81 00
  74      26 EC          BNE EC to 0362.
  76      3F
  77      02             Abort drive motor (safety precaution).
  78      83
  79      7E 02 FD
```

MENU OF ROBOT TASKS (TABLES). (CONSIDERED TO BE IN"RAM" - HENCE CHANGEABLE BY USER).

TABLE 1: TURN HEAD, LIFT ARM, LOWER ARM, TURN HEAD BACK.

```
0380      D0,42          Turn head to 42.
  82      50,20          Lift arm to 20.
  84      50,00          Lower arm to 00.
  86      D0,62          Turn head back to 62.
```

TABLE 2: PIVOT WRIST, ROTATE WRIST, ROTATE AND PIVOT WRIST BACK.

```
0390      90,30          Pivot wrist to 30.
  92      70,00          Rotate wrist to 00.
  94      70,4D          Rotate wrist back to 4D.
  96      90,00          Pivot wrist back to 00.
```

TABLE 3: DRIVE FORWARD, DRIVE BACK.

```
03A0      10,10          Drive forward 10 HEX units.
  A2      14,10          Drive back 10 units.
```

PROCEDURE

1. Execute from 02FD. See "CC" ("chose # cycles").
2. Enter number of cycles of robot action desired. See that number and the prompt "CA" next to it on the display ("CA"="choose action").
3. Enter either 80,90, or A0 for one of 3 possible robot tasks. See number of cycles "CA", and either 80,90, or A0 next to each other on the display. If you entered other than 80,90, or A0 see "CC" and repeat steps 1,2, and 3.
4. The action unfolds. After each cycle, the number of cycles remaining is decremented on the display, but "CA" and 80,90, or A0 remain side by side on the display.

5. When the requested number of cycles have all been executed, the display again showsjust "CC". You again enter the number of cycles desired (see that number and "CA" again) and then the desired task 80,90, or A0. The display will again show the number of cycles, "CA", and either 80,90, or A0, with the cycles of the new robot task unfolding and, as they do, the cycles remaining showing on the left two digits of the display and decrementing as still another cycle is completed.

PROBLEM

Incorporate the abort feature whereby, if switch S3 at port C2A0 on the experimental board is thrown high, the current motor action within the current cycle finishes, the robot re-initializes to its nest, and "CC" shows on the display. See Expt. 8-1 for hints. Also try other "user" tables defining other tasks in the menu at 0380,90, and A0 in "RAM".

CLOSING REMARK

The F3(FC) command can be very powerful in dynamic robot situations. During robot performance possible new sensed values about the environment can be read (input) via a B6 Op Code (LDAA) and then stored in RAM (or EEPROM) tables by means of a B7 Op Code (STAA) to modify the original tables. Next time around (7E 02 FD jump) the new robot action demanded by new environmental conditions will be performed (see Expt. 8-8).

EXPERIMENT 8-4

A TWO-SENSOR APPLICATION: FILLING CARTONS WITH PARTS AS THEY ARRIVE ON A CONVEYOR BELT. CARTON LEAVES WHEN FILLED.

In this application we shall be using two sensors to help implement the robot's "intelligence": its light detector (sensor) and a limit switch S3 on the experimental board (port C2A0). The presence of light at the LDR will inform the robot that a carton has not yet arrived. The robot, in response, is totally inactive. When light is blocked by the arrival of a carton, the robot goes into action to get a part and place it in the carton. It repeats this until the carton is filled to capacity when a limit switch is tripped (we shall simulate that by throwing switch S3 at port C2A0 high). When that switch is tripped the robot is to stop filling the carton, which then moves away allowing light to again be incident on the robot's LDR. The robot then stays idle until the arrival of the next carton blocks light again. A more precise formulation follows.

FORMULATION

1. Robot polls for light. As long as light is detected it remains inactive in its initialized state (head in position 62).

2. When a no light situation is detected (carton has arrived), robot is to go to the parts bin (head turns to 52) to take a part (extend arm, open gripper, close

gripper, retract arm).

3. Robot then goes to the carton (head turns to 72) to place the part in the carton (extends arm, opens gripper, closes gripper, retracts arm). After placement of the part, the robot stays over the carton for, say, 2 seconds (this is an artifice, not at all necessary in real practice. This two second interval will allow us, in carrying out the experiment, the time to simulate a filled carton by throwing S3 high).

4. The robot polls the state of S3 after the above 2 second interval (we've either thrown S3 hi or not in that interval of time). If S3 is found to be lo, the carton is not yet filled with parts and the robot goes back to head position 52 to take another part from the parts bin for placement in the carton at head position 72, as before.

5. If the robot finds S3 to be hi, the carton is filled. The robot goes back to head position 62 (neutral corner) and spends 2 more seconds there before polling the light situation again (as before, this is an artifice to allow us time to bring the light source up to the robot's detector simulating the fact that the filled carton has moved on). We are now back to step 1 above and the chain of events 1-5 above repeats.

```
                POLL FOR LIGHT.  ROBOT INACTIVE IF LIGHT DETECTED (NO CARTON).

0600        3F          Change to R.L.
  01        41          Enable light detector.
  02        83          Change to M.L.
  03        B6 C2 40    LDAA C240.  Read light level.
  06        81 80       CMPA 80.  Light level above 80?
  08        24 F9       BCC F9.  Yes.  No carton present.  Branch to 0603.  Con-
                        tinue to poll for light.

                NO LIGHT.  CARTON ARRIVED.  ROBOT GOES TO BIN TO GET PART.

060A        3F          Change to R.L.
  0B        C3 D0 52    Turn head to 52 (CCW) (parts bin).
  0E        C3 30 20    Extend arm to 20.
  11        C3 A8 30    Open gripper to 30 (slowly).
  14        C3 A8 10    Close gripper to 10 to grasp part.
  17        C3 30 00    Retract arm fully.

                        ROBOT TAKES PART TO PLACE IN CARTON.

061A        C3 D0 72    Turn head CW to 72 (to carton).
  1D        C3 30 20    Extend arm to 20.
  20        C3 A8 30    Open gripper to 30 to release part into carton.
  23        C3 A8 00    Close gripper fully.
  26        C3 30 00    Retract arm fully.

                ROBOT ARM OVER CARTON FOR TWO SECONDS.  POLLS SWITCH S3.
                   IS CARTON FILLED OR NOT?  TAKES APPROPRIATE ACTION.

0629        8F 00 20    Pause for 2 seconds.  During this time we throw S3 hi to
                        simulate filled carton or leave lo (not filled).
```

2C	83	Change to M.L.
2D	B6 C2 A0	LDAA C2A0. Read S3 into ACCA.
30	85 08	BITA 08. Test bit 3 in ACCA i.e. S3.
32	27 D6	BEQ D6. S3 lo (carton not filled). Branch back to 060A to continue fetching and placing a part in the carton.
34	3F	Change to R.L. S3 hi. Carton filled.
35	C3 D0 62	Send head back to 62 (idle position).
38	8F 00 20	Pause for 2 seconds. Allows us time to bring up light source to robot simulating absence of carton. Filled one moved away.
3B	83	Change to M.L.
3C	7E 06 00	JMP 0600 and poll for arrival of a new carton to be filled.

PROCEDURE

1. Execute with light source incident on the robot's LDR. No robot action (no carton present).
2. Remove light source (carton has arrived). Robot goes to head position 52 (bin), gets part, goes to head position 72 and places same in carton. Two second pause.
3. If S3 lo at end of 2 seconds, another part from bin at head position 52 is fetched and placed in the carton at 72. This continues till S3 is hi (filled carton).
4. If S3 hi at end of 2 second pause in (2) above, robot goes to head position 62 (idle) and waits two seconds before it polls for light. If it finds light (carton gone) it remains idle until no light detected (new carton arrived) when (2),(3),(4) repeat.

EXPERIMENT 8-5

SIMULATING CAMERA VISION DETECTION OF THE LOCATION OF A PART: PART PICKED FROM PIECE, TREATED, PLACED BACK ON PIECE. FINISHED PIECE PLACED ON CONVEYOR BELT. REPEAT.

FORMULATION

1. Robot's head position is initially 62 (neutral idle position).

2. Camera vision locates part on a piece which is to be removed from the piece, treated, and placed back on the piece. We simulate the part's location X,Y by entering the head position byte XY (where the part is to be found) from the keyboard (press keys X,Y).

3. Head then goes to that position XY. It extends arm, opens and closes its gripper, and retracts arm ~~to get the part there~~ partially with the part.

4. Head carries the part from head position XY to head position 82 to treat the part there: it extends arm, pauses 2 seconds, and retracts the arm at head position 82 simulating treatment.

5. The head then carries the treated part back to head position XY to place it on the piece (extends arm, pauses 2 seconds to put the part in place).

6. The gripper then opens and closes partially, grasping the whole piece with the part in place. The arm then fully retracts.

7. Head then goes from position XY to position 52 where the robot opens and completely closes its gripper releasing the piece to the conveyor belt.

8. The head then returns to idle or neutral position 62 where it awaits another keyboard input XY simulating camera vision location of a part on the next piece that arrives. The above steps are to then repeat.

The program implementing the above follows. Note the use of the F3 MMMM "motor move" command in which the motor select (SS: here the head) and position (XX) bytes are fetched from addresses MMMM and MMMM+1 respectively. Those bytes are D0 (head move) and XY (position byte entered from the keyboard and simulating camera vision location of the part). D0 is passed to MMMM=0780 by the program itself, while XY is passed to MMMM+1=0781 after XY is input from the keyboard simulating camera vision detection of the position on the piece of the part to be treated.

```
                PARAMETER SET-UP:  PASS "HEAD ROTATE" BYTE (SS) AND "DETECTED" LOCATION
                      (X,Y) OF THE PART (KEYBOARD SIMULATION OF CAMERA VISION).

0700    86 D0       LDAA D0.
  02    B7 07 80    STAA 0780.  Pass D0 to 0780.  D0 represents "head rotate,
                    medium speed" (SS byte).
  05    BD F7 96    JSR IHB.  Enter two keys X,Y.  Byte XY is formed in ACCA
                    and shown on the display.  XY will represent position
                    where the part on the piece was found by "camera vision"
                    and where head must rotate to get it.
  08    B7 07 81    STAA 0781.  Pass XY location of part to 0781.  Will become
                    position to which head will rotate (XX byte in motor move)

                HEAD MOVEMENT TO PART LOCATION XY.  TAKING OF THE PART.

070B    3F          Change to R.L.
  0C    F3 07 80    Motor move, wait, abs, extended.  F3 fetches SS byte from
                    0780 and XX byte from 0781 to cause head to turn to position
                    XY to get the part.
  0F    C3 30 20    Extend arm to 20.
  12    C3 A8 30    Open gripper to 30.
  15    C3 A8 10    Close gripper to 10 (grasp part).
  18    C3 30 00    Retract arm fully.

                TAKE PART TO HEAD POSITION 82 FOR TREATMENT.

071B    C3 D0 82    Move head to 82.
  1E    C3 30 20    Extend arm to 20 to "dip the part".
  21    8F 00 20    2 second pause while part is treated.
  24    C3 30 00    Retract arm fully.
```

GO BACK TO HEAD POSITION XY WITH TREATED PART. PUT BACK INTO THE PIECE. GRASP THE WHOLE PIECE.

```
0727    F3 07 80    Head goes back to position XY.
  2A    C3 30 20    Extend arm to 20.
  2D    8F 00 20    Wait 2 seconds (part is being put back in place on the
                    piece).
  30    C3 A8 50    Open gripper to 50.
  33    C3 A8 40    Close gripper to 40 i.e. gripper grasps the whole piece.
  36    C3 30 00    Retract arm fully.
```

TAKE PIECE TO CONVEYOR BELT (HEAD POSITION 52). PLACE PIECE ON BELT.

```
0739    C3 D0 52    Move head to position 52 (conveyor belt).
  3C    C3 A8 50    Open gripper to 50 releasing the piece.
  3F    C3 A8 00    Close gripper completely.
```

HEAD RETURNS TO NEUTRAL POSITION 62 TO AWAIT "CAMERA VISION" INPUT OF LOCATION OF PART ON NEXT PIECE. REPLAY OF ALL THE ABOVE EVENTS.

```
0742    C3 D0 62    Head goes to 62 to wait for new part position information.
  45    83          Change to R.L.
  46    BD F6 4E    JSR REDIS.
  49    7E 07 00    JMP 0700. Jump to 0700 to await input of next part position.
                    Sequence of events will repeat.
```

PROCEDURE

Follow the formulation section as a precise guide to the procedure. As a start, enter 92 after the program is executed. You'll see 92 on the display, the head go from neutral 62 to 92, the arm extend, the gripper open and close to get the part, the arm retract, the robot taking the part to 82 for treatment, the arm extend to treat the part, a pause of 2 seconds while the part is being treated, the arm retract, taking of the part back to 92, extending of the arm to place the part on the piece, another wait of 2 seconds as the part is being placed, opening of the gripper to grasp the whole piece, closing of the gripper partially to take the piece, head rotating to 52 (location of conveyor belt), opening of the gripper to release the piece to the belt, closing of the gripper (fully) and return of head to neutral position 62 to await keyboard input of the location of the part on the next piece. The above events will then repeat. Try 72 as the next part position location.

EXPERIMENT 8-6

A BUSY ROBOT PLAYS CATCH-UP WITH A MOVING CONVEYOR BELT TO PLACE THE PART.

FORMULATION

Consider a robot busy at some activity concerned with a part or piece. At the

end of its activity it starts to move toward a point on a conveyor belt that has been moving while the robot was busy. In the process of going to the point it trips a switch which is connected to the IRQ interrupt line of the CPU. We shall simulate that by throwing the INT switch we have provided on our experimental board (it is one of the 8 possible interrupt sources at inport C200 which is NANDed to the IRQ line - see Expt. 3-4). All the while the robot was busy, it was keeping track of time (and, therefore, of distance the conveyor belt traveled from the start of its activity). The onset of the interrupt should provide the means for retrieving the elapsed time (which has been updated and stored in some memory location as time progressed). That elapsed time, when multiplied by a constant factor (the speed of the conveyor belt) should tell us exactly how far the conveyor belt traveled. We can then have the robot's drive motor send the robot forward that distance to catch up with the point on the conveyor belt where the robot is to place the completed part or piece. We shall be neglecting the further distance the belt travels while the robot is moving to catch up with the point on the belt where it is to place the part. Allowance can be made for that if we knew the speed of robot movement relative to that of the conveyor belt and we leave it to you as a problem to make the necessary modification of our program below. For our purposes here, we shall assume the robot moves much faster than the slowly moving conveyor belt (or you might think of the interrupt routine as also stopping the conveyor belt to wait for the robot to overtake the point at which it is to place the part).

Our method of counting elapsed time while the robot is busy will involve incrementing a given location in memory every 1/25 sec. and then, at the time of the interrupt, retrieving the contents of that memory location (and, optionally, stopping the conveyor belt). Multiplying those contents by the conveyor's speed (in our case, dividing it by 4 for convenience of size of numbers) will give the distance the robot must travel to catch up with the fixed point on the belt. That distance will be written into another memory location which, together with the byte 08 passed into the immediately preceding location, will serve as the SS and XX operands for the F3 "motor move, wait, abs. extended" command. 08 selects the drive motor at slow speed in the forward direction. (This experiment will thus also show the power of the F3 command in dynamic situations where motor select and position bytes (SS,XX) can, as here, be determined or input "on the fly" during run time, as distinguished from the case where they are fixed parameters entered at program time. Expt. 8-1 showed the same power for the E3 (or EC) motor move command (indexed addressing). There, motor selects, SS, and associated distances, XX, were entered from the keyboard at run time and later retrieved. Here position XX is being updated in memory and then retrieved).

We shall store the number of 1/25 sec. intervals at address 0490 and that number divided by 4 (i.e. distance XX) at address 0461. The motor select byte SS=08 (drive motor) will be passed into address 0460 so that the motor move command F3 0460, when invoked, will cause the drive motor to go forward at slow speed a distance given by the byte XX found in address 0461. The storing of distance at 0461 and the invoking of F3 0460 will both be done in the interrupt routine which we shall write at 0420. Remember, too, that an interrupt IRQ (INT) request from the experimental board causes vector entry at 002D where we have three bytes available to jump ourselves to the interrupt routine (7E 04 20 will do that). Hence we must have the program itself write the 7E,04,20 bytes into locations 002D,2E, and 2F.

INITIALIZATION AND COUNT-UP OF TIME IN 1/25 SEC. INTERVALS.

```
0400    0E          CLI. Enable IRQ interrupt line.
  01    8E 0E D0    LDS 0ED0. Set stack pointer.
  04    86 7E       LDAA 7E.
  06    B7 00 2D    STAA 002D. 7E to 002D.
  09    CE 04 20    LDX 0420. 0420 to X reg.
  0C    DF 2E       STX 2E. 04 to 002E and 20 to 002F (direct addressing).
  0E    7F 04 90    CLR 0490. 0490 will hold the number of 1/25 sec. time
                    intervals elapsed.
  11    CE 0A FF    LDX 0AFF for 1/25 sec. time delay.
  14    09          DEX.
  15    26 FD       BNE FD back to 0414.
  17    7C 04 90    INC 0490. Another 1/25 sec. interval has elapsed. BRA F5
  1A    20 F5       back to 0411 to do and record another 1/25 sec. interval.
```

INTERRUPT ROUTINE. DRIVE ROBOT FORWARD TO CATCH UP WITH POINT ON CONVEYOR BELT.

```
0420    B6 04 90    LDAA 0490. Read number of elapsed 1/25 sec. intervals
                    into ACCA.
  23    44          LSRA. Logic shift right on ACCA. Divides by 2.
  24    44          LSRA. Divide again by 2. (ACCA) now distance.
  25    B7 04 61    STAA 0461. Store distance conveyor traveled at 0461.
  28    86 08       LDAA 08.
  2A    B7 04 60    STAA 0460. Store 08 (drive motor select byte SS) at 0460.
  2D    3F          Change to R.L.
  2E    F3 04 60    Move robot forward the distance stored in 0461.
  31    02          Abort drive (safety precaution).
  32    83          Change to M.L.
  33    20 FE       BRA FE. Halt.
```

The following table will be helpful in analyzing the results to be expected when you execute the above program.

TABLE OF EXPECTED VALUES

TIME ELAPSED (SEC)	CONTENTS OF LOCATION 0490. DECIMAL NUMBER OF 1/25 SEC. INTERVALS	CONTENTS OF LOCATION 0461. (0490)/4 IN HEX. CATCH-UP DISTANCE BYTE
1	25	06
2	50	0C
3	75	13
4	100	19
5	125	1F
6	150	26
7	175	2C
8	200	32
9	225	38
10	250	3E

PROCEDURE

Execute the program and keep track of time with a stop watch. At, say, the end of 2 seconds, create an interrupt with INT on the board. You should see the robot go forward a certain distance corresponding to the distance byte 0C in the above table. Measure that distance. Re-run the program and this time create the interrupt at the end of 4 seconds. The robot will go forward a distance corresponding to the byte 19 in the table. Measure that distance. It should be double the distance observed when the interrupt was made at the end of 2 seconds. Finally, after again executing the program, cause the interrupt to occur at the end of 8 seconds. The robot will go forward a distance corresponding to the byte 32 in the table. Measure the distance. You should find it double the distance previously traveled when the interrupt was made at the end of 4 seconds. In each case examine memory location 0461 after the robot comes to a halt. Its contents should agree with the last column in the table corresponding to the time elapsed till the interrupt was made.

PROBLEM

Incorporate in the program the possibility that the conveyor belt is moving while the robot is trying to catch up to it. Assume relative speeds and work out the correction factor i.e. the added distance the point on the belt travels while the robot is overtaking it. The robot must go that added distance. Also work out the other possibility that the interrupt routine stops the conveyor belt so no correction factor (added distance) need be taken into account

EXPERIMENT 8-7

PLACING PARTS FROM TWO BINS ON TO AN ASSEMBLY; TESTING FOR EMPTY BINS (FINISHED PIECE); QUALITY CONTROL TEST OF THE FINISHED PIECE (REJECT IT OR PASS ON); DISPLAY OF NUMBER OF GOOD vs. BAD PIECES DURING PRODUCTION.

This experiment would have wide applicability in the world of manufacturing operations. The formulation of the problem below describes and defines the various aspects of the manufacturing operation we have in mind.

FORMULATION

1. With the base of the robot at the origin, head position 52 will define the location of bin 1 containing a number of parts which are to be placed on the piece being assembled. Head position 42 will define the location of bin 2 containing a number of other parts which, after bin 1 is empty, must be placed on the piece being assembled. The piece being assembled is at head position 62.

2. An empty bin will be detected if the robot senses light (think of a light beam at the bottom of the bin being blocked as long as there is a single part left in the bin).

3. Sensing that both bins are empty will signify that the piece being assembled is finished. The robot must then go to head position 62 to pick it up.

4. The robot base is to then drive forward with the assembled piece a fixed distance to a quality control test station where a 3 or 4 sec. QC test is undertaken. If at the end of that test the piece is found good it is placed in the "accept" bin which is at head position 72 in this new robot base position. The display updates the "good pieces" count. If, however, the QC test shows that the piece fails, it is placed in the "reject" bin which is at head position 52 in this new robot forward position and the "bad pieces" count is updated. The QC test result (pass or fail) will depend on whether a switch is thrown high (fail) or remains low (pass). (Think of the QC test as determining whether e.g. a size tolerance "spec" is met - size too large and switch is tripped, otherwise not). We shall simulate that test with our switch S3 on the experimental board (inport C2A0). Thus at any time the display will read out the number of good vs. bad assemblies produced so far.

5. After the "accept" or "reject" operation in (4), the head turns to 52 (it's already there in the reject case) and the robot drives back to the original base position facing bin 1 (position 52) and is ready to take parts again from bin 1 and then from bin 2 and place them on to a new piece being assembled at 62. New assembly will commence only if the light sensing test indicates absence of light in the bins i.e. parts are present.

In the program that follows, we shall not incorporate the details of arm extending into and out of bins, with gripper opening and closing to take a part, or arm and gripper doing the same to take the assembled piece etc. We leave it to you to add all those desirable features - they are details. We are after the essentials of the method, the application, and the program. Those details are easily incorporated into the program using the appropriate motor move commands C3 SS XX as we have been doing throug-out the book. You are urged to incorporate those details after you have performed and understood the experiment as formulated above and programmed below.

INITIALIZE COUNTERS OF GOOD AND BAD FINISHED PIECES. DISPLAY 00,00.

```
0300      7F 03 C0      CLR 03C0.  Location 03C0 will count good pieces.
  03      7F 03 C1      CLR 03C1.  Location 03C1 counts bad pieces.
  06      BD F6 4E      JSR REDIS.
  09      B6 03 C0      LDAA 03C0.  00 read into ACCA.
  0C      BD F7 AD      JSR OUTBYT.  Display 00.
  0F      B6 03 C1      LDAA 03C1.  00 read into ACCA.
  12      BD F7 AD      JSR OUTBYT.  Display 00.
```

GET PARTS FROM BIN 1 AND PLACE ON PIECE AT ASSEMBLY STATION.
POLL BIN 1 FOR LIGHT (EMPTY BIN CONDITION).

```
0315      3F            Change to R.L.
  16      41            Enable light detector.
  17      C3 D0 52      Turn head to position 52.  Bin 1 there.
  1A      8F 00 10      Pause 1 sec. there (simulates taking a part).
  1D      83            Change to M.L.
  1E      B6 C2 40      LDAA C240.  Read light intensity.
  21      81 80         CMPA 80.  Light level greater than 80 (i.e. bin 1 empty)?
```

```
  23      22 0A          BHI 0A. Yes. Branch to 032F. Bin 1 empty. Get parts in
                         bin 2 for placement.
  25      3F             Change to R.L. No. Bin 1 still has parts.
  26      C3 D0 62       Turn head to 62 to place part on piece there.
  29      8F 00 10       Pause 1 sec. there (simulates placing part on the piece).
  2C      83             Change to M.L.
  2D      20 E6          BRA E6 back to 0315 to get another part from bin 1.

     BIN 1 EMPTY (LIGHT DETECTED). GET PARTS FROM BIN 2 AND PLACE ON PIECE
        AT ASSEMBLY STATION. POLL BIN 2 FOR LIGHT (EMPTY BIN CONDITION).

032F      3F             Change to R.L.
  30      C3 D0 42       Turn head to bin 2 at head position 42.
  33      8F 00 10       Pause 1 sec. at bin 2 (simulates taking part there).
  36      83             Change to M.L.
  37      B6 C2 40       LDAA C240. Read light level.
  3A      81 80          CMPA 80. Light level greater than 80 (i.e. bin 2 empty)?
  3C      22 0A          BHI 0A. Yes. Branch to 0348. Bin 2 empty. Take assembled
                         piece at 0348 for QC test.
  3E      3F             Change to R.L. No. Bin 2 still has parts.
  3F      C3 D0 62       Turn head to 62 to place part on piece there.
  42      8F 00 10       Pause 1 sec. there (simulates placing part on the piece).
  45      83             Change to M.L.
  46      20 E7          BRA E7 back to 032F to get another part from bin 2.

     BINS 1 AND 2 ARE EMPTY. PIECE HAS BEEN ASSEMBLED. GET THE PIECE AND TAKE
           IT TO THE QUALITY CONTROL TEST STATION (DRIVE FORWARD).

0348      3F             Change to R.L.
  49      C3 D0 62       Turn head to assembly station to get finished piece.
  4C      8F 00 10       Pause 1 sec. (simulates taking the piece).
  4F      C3 08 10       Go forward 10H units to QC test station.
  52      02             Abort drive motor (safety precaution).

                    A 3 OR 4 SECOND QUALITY CONTROL TEST.

0353      83             Change to M.L.
  54      C6 03          LDAB 03 for time delay.
  56      CE FF FF       LDX FFFF for time delay.
  59      B6 C2 A0       LDAA C2A0. Read switch S3. S3 hi means QC test failed.
  5C      85 08          BITA 08. Test S3 for hi or lo.
  5E      26 31          BNE 31. S3 hi, bad piece. Branch to 0391 to place bad
                         piece in reject bin.
  60      09             DEX. So far piece is good. Continue with 3 or 4 sec. t.d.
                         and continue QC testing.
  61      26 F6          BNE F6. Branch back to 0359 to poll S3 anew.
  63      5A             DECB. X=00. Decrement B.
  64      26 F0          BNE F0. If B≠0 branch back to 0356 and do time again.
                         Piece still good.
```

QC TEST PASSED. UPDATE GOOD/BAD PIECE COUNT ON THE DISPLAY. PLACE GOOD NEW PIECE IN "ACCEPT" BIN, DRIVE BACK TO BIN 1 TO TAKE PARTS FOR A NEW PIECE TO BE ASSEMBLED (IF STATE OF LIGHT SENSOR INDICATES NO LIGHT).

```
0366    7C 03 C0    INC 03C0 (number of good pieces).
  69    BD F6 4E    JSR REDIS.
  6C    B6 03 C0    LDAA 03C0. Read updated number good pieces.
  6F    BD F7 AD    JSR OUTBYT. Display same.
  72    B6 03 C1    LDAA 03C1. Read updated number bad pieces.
  75    BD F7 AD    JSR OUTBYT. Display same
  78    3F          Change to R.L.
  79    C3 D0 72    Turn head to 72 to "accept" bin.
  7C    8F 00 10    Pause 1 sec. Simulates placing good piece in "accept" bin.
  7F    C3 D0 52    Turn head to 52 (bin 1's location after robot drives back).
  82    C3 0C 10    Drive back 10H to bin 1.
  85    02          Abort drive (safety precaution).
  86    83          Change to M.L.
  87    B6 C2 40    Read light level in bin 1.
  8A    81 80       CMPA 80. Light level above 80?
  8C    24 F9       BCC F9. Yes. No parts in bin 1 or 2. Branch back to 0387
                    and keep polling for parts.
  8E    7E 03 15    JMP 0315. Parts available in bin 1 and bin 2 (no light).
                    Take new parts from bin 1 at 0315 to place on a new
                    piece being assembled.
```

QC TEST FAILED. UPDATE GOOD/BAD PIECE COUNT ON THE DISPLAY. PLACE BAD PIECE IN "REJECT" BIN. DRIVE BACK TO BIN 1 TO TAKE PARTS TO A NEW PIECE TO BE ASSEMBLED (IF NO LIGHT DETECTED FROM LIGHT SENSOR).

```
0391    7C 03 C1    INC 03C1 (number of bad pieces).
  94    BD F6 4E    JSR REDIS
  97    B6 03 C0    Read updated number good pieces.
  9A    BD F7 AD    JSR OUTBYT. Display same.
  9D    B6 03 C1    Read updated number bad pieces.
  A0    BD F7 AD    JSR OUTBYT. Display same.
  A3    3F          Change to R.L.
  A4    C3 D0 52    Turn head to 52 to "reject" bin.
  A7    8F 00 10    Pause 1 sec. (simulates placing bad piece in "reject" bin).
  AA    C3 0C 10    Drive back 10H to bin 1 (head is in correct position 52).
  AD    02          Abort drive. Safety precaution.
  AE    83          Change to M.L.
  AF    B6 C2 40    Read light level in bin 1.
  B2    81 80       CMPA 80. Light level above 80?
  B4    24 F9       BCC F9. Yes. No parts in bin 1. Branch back to 03AF to
                    keep polling.
  B6    7E 03 15    JMP 0315. Parts are available in bins 1 and 2 (no light).
                    Jump to 0315 to start taking new parts from bin 1 for new
                    new piece being assembled.
```

PROCEDURE AND OBSERVATIONS

1. Execute the program with no light incident on the sensor i.e. bin 1 has parts. See 0000 on the display, indicating 00 good and 00 bad pieces produced so far.

Observe the head go from position 62 to 52, spend 1 sec. there "taking a part" from bin 1, go back to 62, spend 1 sec. there "placing the part" on the piece being assembled, go back to 52 to take another part from bin 1, go back to 62 to place it on the piece, etc.

2. Direct a light source at the robot eye when the head is at bin 1 (making sure the light is still incident at the moment the 1 sec. pause of the head there is over). This tells the robot there are no parts left in bin 1. The head now goes to bin 2 at position 42.

3. The robot "takes parts" from bin 2 at head position 42 and brings each one to the piece at the assembly station at head position 62, going back and forth between 42 and 62 and spending 1 sec. at each position to simulate taking and placing of parts.

4. At any time of your choosing, direct the light source at the robot's eye simulating an empty bin 2 (light should be present after the 1 second pause at position 42-bin 2). The head now goes back to 62 to "take the finished piece", spending 1 sec. there simulating taking of the piece.

5. The robot then drives forward 10H units, stops, spends 3 secs there simulating a QC test (actually polling switch S3 on the experimental board for hi or lo - hi indicating failure).

6. If the robot finds S3 lo for 3 secs, the piece has passed the test. The display changes to 0100 indicating 1 good, 0 bad pieces produced. Robot turns its head to position 72 to face the "accept" bin, spends 1 sec. there "placing the good piece", turns its head back to 52 (to face bin 1 when it drives back).

7. If no light is at the robot's eye, bins 1 and 2 have parts that are again taken from bin 1 (after the 1 sec. pause there) to the assembly station at 62 and all the above steps and observations are repeated for the assembly of a new piece from the parts in bins 1 and 2. If, however, light is at the robot's eye, nothing happens (bins are empty) until there is no light indicating bins 1 and 2 have parts. The sequence of events repeats.

8. In step 5 above (QC test), next time around, throw S3 hi during the 3 sec. delay. This simulates a failure of the piece produced. The display will now show 0101 (1 good, 1 bad piece produced so far). Head immediately turns to 52 to face the "reject" bin and spends 1 sec. there simulating placing of the bad piece. The robot then drives back 10H units and is in the correct head position 52 to face bin 1. If light is at the robot's eye (bins 1 and 2 still empty) nothing further will occur. If there is no light (or it is removed) the bins have parts and we are back into production of a new piece at the assembly station.

9. Do this a number of times, simulating good and bad pieces using S3 during QC and note how the display always shows the updated good/bad piece count.

PROBLEM

Modify the program to include all details making the experiment as realistic as possible, i.e. extend/retract/raise/lower arm, open/close gripper, replace 1 sec. pauses by actual activities, etc.

EXPERIMENT 8-8

A ROBOT LEARNS FROM ITS ENVIRONMENT AS IT DOES ITS TASK (REPROGRAMS ITSELF "ON THE FLY" - IF REQUIRED - AND DOES TASK DIFFERENTLY NEXT TIME AROUND).

FORMULATION

1. Our "task" will consist of the robot turning its head from 62 to 42 (CCW), then lifting its arm from 00 to 20, then lowering it from 20 to 00, then turning its head back from 42 to 62. The task then repeats indefinitely.

2. Should the robot ever meet light as its head turns from 62 to 42, then the task will be modified next time around. It will "short circuit" the "arm lift and lower" actions and, henceforth, the task will only consist of a head rotate to 42 then back to 62, then back to 42, etc. i.e. the robot will have "learned from its environment".

3. The original task will be defined by a main program and a table of motor bytes in which the F3(FC) "motor move, wait (continue), abs., extended" commands are employed (see Expt. 8-3, "A Menu of Robot Tasks").

4. You are to think of the main "driver" program as residing in ROM (as it most likely would in practice) and the "task bytes" as residing in a table in RAM (which the user has set up). These "task bytes" consist of 2 byte pairs (SS for motor/speed select and XX for absolute destination position). Again, reference Expt. 8-3 for discussion on the F3(FC) commands.

5. When we say in (2) above that the task will be modified if the head encounters light as it goes from head position 62 to 42, we are saying that our program below will alter the "task byte table" in RAM if light is found but leave it alone if light is not detected. In our example, the alteration will consist of "obliterating" the "arm lift - arm lower" portion of the original task so that the next time the task is performed it will consist only of head rotate CCW/CW, without the arm lift/lower portion. The main program follows.

DRIVER FOR THE ORIGINAL TASK (IN "ROM"): HEAD ROTATES, ARM LIFTS, ARM LOWERS, HEAD ROTATES BACK, ETC.

```
0900    3F          Change to R.L.
  01    41          Enable light detector
  02    FC 09 80    Motor move, abs, continue, extended. Executes D0,42
                    bytes at 0980,81 (rotate head CW to 42).
  05    83          Change to M.L.
  06    B6 C2 40    LDAA C240. Read light sensor.
  09    81 E0       CMPA E0. Light greater than E0?
  0B    22 10       BHI 10. Yes. Branch to 091D to change task table.
  0D    3F          Change to R.L. No. Don't change table.
  0E    1E F5       "Branch if arm(head) busy" back to 0905 and poll for light
                    as head moves to 42.
  10    F3 09 82    Head reached 42 (not busy). Pick up motor bytes 50,20 at
                    0982,83 (lift arm to 20H and wait till completed).
```

```
  13      F3 09 84      Arm reached 20. Execute motor bytes at 0984,85(50,00) to
                        lower arm to 00. Wait till completed.
  16      F3 09 86      Arm reached 00. Execute motor bytes at 0986,87(D0,62). Ro-
                        tates head CW back to 62.
  19      83            Change to M.L.
  1A      7E 09 00      Task done. JMP 0900 to do it again.

          ROBOT HEAD FOUND LIGHT ON WAY FROM 62 TO 42. CHANGE TASK
              TABLE AT 0980 IN "RAM". TASK WILL HENCEFORTH BE
                        DIFFERENT (NO ARM RISE OR LOWER).

091D      86 D0         LDAA D0 (D0=head rotate SS byte).
  1F      B7 09 82      STAA 0982. Write D0 to 0982.
  22      B7 09 84      STAA 0984. D0 to 0984.
  25      86 42         LDAA 42 (42=head position XX)
  27      B7 09 83      STAA 0983. 42 to 0983.
  2A      B7 09 85      STAA 0985. 42 to 0985.
  2D      7E 09 0D      JMP 090D. Continue head move from 62 to 42 at 090D.

        THE MOTOR BYTE "TASK TABLE" IN RAM (SS,XX=MOTOR SELECT, POSITION).

0980      D0,42         Head rotate CCW to 42, medium speed.
  82      50,20         Arm lift to 20, medium speed.
  84      50,00         Arm lower to 00, medium speed.
  86      D0 62         Head rotate CW to 62, medium speed.
```

PROCEDURE

Execute the program. See the main task unfold (head rotate from 62 to 42, arm lift to 20, arm lower to 00, head rotate back to 62) and repeat over and over. Direct light at the robot eye sometime when the head is rotating from 62 to 42. Observe that the very next time the task repeats, the lifting and lowering of arm are no longer executed. The modified tasks now consist only of head rotates 62 to 42 to 62 etc. The robot reprogrammed itself to do a modified task based on a new environmental condition (here presence of light). Reset and check the bytes at 0980 through 0987. What do you find? Note how our main program at 091D-092F changes the task table to D0,42 at 0982,83 and again at 0984,85 thereby "pinning the head" to 42 ("short circuiting" arm lift and lower actions previously there) till it is made to move to 62.

Can you see how EEPROMs (Electrically, Erasable, Programmable, Read Only Memories) will play an important role in robot technology?

EXPERIMENT 8-9

A ROBOT SLEEPS OR WORKS, ANNOUNCES "LUNCH TIME", GOES TO LUNCH, ASKS FOR "FOOD", TAKES IT, "FEEDS SELF", COMMENTS ON IT, RETURNS TO HOME BASE TO RESUME SLEEP OR WORK. ETC.

FORMULATION

1. When the program executes, the operator is to enter either key 0 (sleep command) or key 1 (work command).

2. If sleep command is entered, operator must follow with the press of a key (1, 2,...or F) for the number of "snores" allowed (length of sleep time). If the work command is entered, operator must follow with the number of work cycles desired (entered by pressing any key from 1 to F). A work cycle will consist of the gripper opening and partially closing (pick operation), followed by a head rotate to 52 (to place), followed by gripper opening and completely closing, followed by a head rotate back to initial position 62 (to pick again).

3. When either the sleep period or work cycles are over, robot says "lunch time, lunch time" and moves forward 20H units. It then says "food, food" and pivots wrist down and under.

4. Robot then swings head around to position 42, opens gripper, extends arm, partially closes gripper (around a cup), retracts arm (with cup), sends arm partially up, pivots wrist farther up, sends arm up further, pivots wrist up some more, spends 3 seconds, or so, "drinking", says "good, good", pivots wrist a bit down, sends arm part way down, pivots wrist further down, sends arm all the way down, extends arm, opens gripper (releasing cup to floor), retracts arm, closes gripper, turns head back to initial 62 position, pivots wrist back down and under to its original 00 position, drives back to "home" or "office" and says "ready".

5. Operator then enters key 0 or key 1 for sleep or work mode followed by the number of "snores" or work cycles desired, as the case may be. The sequence of events repeats.

The program to accomplish this is straightforward and not at all difficult.

NOTE

Our HERO is "right-armed" i.e. looking down from the top, with the keyboard in front of you, the arm swings up to the left. If yours is "left-armed" i.e. arm swings up to the right, you'll have to modify certain parts of the program to conform with the above formulation. See problem later.

ENTER KEY 0 OR KEY 1 FOR SLEEP OR WORK MODE.

```
0400    BD F7 77    JSR INCH. Enter key 0 (sleep) or key 1 (work).
  03    81 00       CMPA 00. Key 0 (sleep)?
  05    27 06       BEQ 06. Yes. Branch to 040D (sleep).
  07    81 01       CMPA 01. No. Key 1 (work)?
  09    27 0F       BEQ 0F. Yes. Branch to 041A (work).
```

```
 0B      20 FE        BRA F3.  Neither.  Branch to 0400.  Try again.

                              SLEEP MODE

040D     BD F7 77     JSR INCH.  Key in number of snores allowed (1,2,...,F).
                      Save in (ACCA).
  10     3F           Change to R.L.
  11     72 04 D0     "Speak" the snore phonemes at 04D0.
  14     83           Change to M.L.
  15     4A           DECA.  Another snore done.
  16     26 F8        BNE F8 to 0410 and do another snore.
  18     20 1B        BRA 1B.  All snores done.  Branch to 0435 to wake up and say
                      "lunch time".

                              WORK MODE

041A     BD F7 77     JSR INCH.  Key in number of work cycles desired (1,2,...F).
                      Save in ACCA.
  1D     16           TAB.  Save in ACCB.
  1E     3F           Change to R.L.
  1F     C3 A8 20     Open gripper to take part.
  22     C3 A8 10     Close gripper partially to grab the part.
  25     C3 D0 52     Turn head to 52.
  28     C3 A8 20     Open gripper to release part.
  2B     C3 A8 00     Close gripper completely.
  2E     C3 D0 62     Turn head back to 62.
  31     83           Change to M.L.
  32     5A           DECB.  Another work cycle done.
  33     26 E9        BNE E9 back to 041E to do another work cycle.  Cycles not
                      done yet.

          SLEEP OR WORK OVER.  ROBOT SAYS "LUNCH TIME, LUNCH TIME".

0435     3F           Change to R.L.
  36     72 04 A0     Speak phonemes at 04A0.
  39     72 04 A0     Speak them again.

              ROBOT DRIVES TO LUNCH AND SAYS "FOOD, FOOD".

043C     D3 10 20     Move forward 20 units.
  3F     72 04 B0     Speak phonemes at 04B0.
  42     72 04 B0     Repeat them.

          ROBOT TAKES FOOD, "FEEDS SELF", COMMENTS ON THE FOOD.

0445     C3 90 98     Pivot wrist down and under to 98.
  48     C3 D0 28     Rotat head to 28.
  4B     C3 A8 70     Open gripper.
  4E     C3 30 50     Extend arm to 50.
  51     C3 A8 35     Close gripper around cup.
  54     C3 30 00     Retract arm (with cup).
  57     C3 50 30     Swing arm to 30.
  5A     C3 90 0D     Pivot wrist from 98 to 9D.
  5D     C3 50 50     Swing arm from 30 to 50.
```

```
  60      C3 90 A5      Pivot wrist from 9D to A5.
  63      8F 00 30      Pause 3 secs. to "drink" out of cup.
  66      72 04 C0      Speak "good" phonemes at 04C0.
  69      72 04 C0      Repeat them.
  6C      C3 90 9D      Pivot wrist from A5 to 9D.
  6F      C3 50 30      Send arm down from 50 to 30.
  72      C3 90 98      Pivot wrist from 9D to 98.
  75      C3 50 00      Swing arm down from 30 to 00 (vertical).
  78      C3 30 50      Extend arm to 50.
  7B      C3 A8 70      Open gripper releasing cup to floor.
  7E      C3 30,00      Retract arm (without cup).
  81      C3 A8 00      Close gripper completely.
  84      C3 D0 62      Rotate head back to 62.
  87      C3 90 00      Pivot wrist down and under back to 00.

                  LUNCH OVER.  DRIVE BACK.  SAY "READY".

048A      D3 14 20      Drive back.
  8D      72 F4 A1      Say "ready" (phonemes at F4A1 in monitor).

       ROBOT WAITS FOR OPERATOR TO PRESS SLEEP (0) OR WORK (1) KEY AGAIN.

0490      83            Change to M.L.
  91      7E 04 00      JMP 0400.  Start all over.

                       PHONEMES FOR "LUNCH TIME".

04A0-AF:  18,32,31,0D,2A,10,3E; 2A,15,00,29,0C,3E,3E,3E; FF (return).

                            PHONEMES FOR "FOOD".

04B0-B9:  1D,37,37,37,37,1E,3E,3E,3E; FF (return).

                            PHONEMES FOR "GOOD"

04C0-C9:  1C,36,36,36,36,1E,3E,3E,3E; FF (return).

                      PHONEMES FOR SNORING (ZH,ZH,ZH,...).

04D0-EA:  07,07,07,07,07,07,07,07,07,07,07,07,07,07,07;
          3E,3E,3E,3E,3E,3E,3E,3E,3E,3E,3E; FF (return).
```

Run the program and do exactly as indicated in the formulation section of this experiment.

PROBLEM

Add other interesting features of your own to the experiment (bar hopping, tipping, flirting, etc.). For those with a "left-armed" robot make the necessary changes for the self-feed part of the program (don't look, but the answer should be inserting 00 at 0447, 045C,0462,046E, and 0474 as one possibility).

EXPERIMENT 8-10

HERO WELCOMES PARTICIPANTS AT A SEMINAR IN ROBOTICS. SERVES THEM.

This experiment might be a fun way to introduce HERO to the participants of a robotics seminar. The author is grateful to Western Electric Co. (Indianapolis) and Mr. Dave Donville for their permission to use the latter's name in HERO's speech in this experiment.

1. HERO says "Welcome to robotics and Donville's doughnuts. Ask me for a cup for coffee. Dave serves doughnuts. I repeat: welcome to robotics...
2. Participant comes over to the robot and says: "cup, please".
3. After 1 second pause, robot says "O.K.".
4. Robot then goes forward 3 or 4 feet.
5. Robot pivots wrist down and under.
6. Robot rotates head to location of cup on floor.
7. Robot opens gripper.
8. Robot lowers arm (gripper encircles cup).
9. Robot tightens gripper around cup.
10. Robot retracts arm with cup.
11. Robot turns head back to original point.
12. Robot drives back, retracing steps.
13. Robot swings arm up to horizontal.
14. Robot pivots wrist back so cup is vertical.
15. Robot say "press key, take cup".
16. Participant presses any key.
17. Gripper opens releasing cup into participant's hands.
18. Gripper closes.
19. Wrist pivots back down and under.
20. Arm swings back down.
21. After 1 sec. pause, robot says "next". Next participant requests "cup please."
22. The sequence of events repeats.

The program is straight-forward. Again note that our program pertains to our "right-armed" HERO (see note, Expt. 8-9). Modify accordingly if yours is "left-armed".

OPENING SPEECH

```
0200      72 02 80      Speak speech phonemes at 0280.
  03      72 02 F0      Speak phonemes at 02F0 ("I repeat").
  06      72 02 80      Repeat phonemes at 0280.
```

POLL SOUND DETECTOR FOR PARTICIPANT'S CUP REQUEST.

```
0209      42            Enable sound detector.
  0A      83            Change to M.L.
  0B      B6 C2 40      LDAA C240.  Detect and input sound.
  0E      81 30         CMPA 30.  Sound level above 30?
  10      25 F9         BCS F9.  No.  Branch to 020B (carry set), poll sound.
```

PARTICIPANT SAID "CUP PLEASE". ROBOT RESPONDS AND GOES TO GET CUP. RETURNS WITH CUP.

```
0212    3F          Change to R.L.
  13    8F 00 10    Pause for 1 sec.
  16    72 02 60    Speak the phrase "O.K."
  19    D3 10 30    Robot drives forward 3 or 4 feet.
  1C    C3 90 98    Robot pivots wrist down and under.
  1F    C3 D0 42    Head rotates to position 42.
  22    C3 A8 70    Open gripper.
  25    C3 30 50    Extend arm to 50 around cup.
  28    C3 A8 35    Close gripper around cup.
  2B    C3 30 00    Retract arm with cup.
  2E    C3 D0 62    Head rotates to 62.
  31    D3 14 30    Drives back with cup.
  34    C3 50 60    Swing arm up to 60.
  37    C3 90 56    Pivots wrist from 98 to 56.  Cup is now vertical.
```

ROBOT SAYS "PRESS KEY, TAKE CUP."

```
023A    72 03 00    Speak phonemes at 0300
```

PARTICIPANT PRESSES KEY, CUP IS RELEASED, ARM AND WRIST GO BACK TO INITIAL POSITIONS.

```
023D    83          Change to M.L.
  3E    B7 F7 77    JSR INCH.  Press any key.
  41    3F          Change to R.L.
  42    C3 A8 70    Open gripper (release cup).
  45    C3 A8 00    Close gripper.
  48    C3 90 00    Pivot wrist down and under.
  4B    C3 50 00    Swing arm down to vertical.
```

PAUSE. ROBOT SAYS "NEXT".

```
024E    8F 00 10    Pause 1 sec.
  51    72 02 70    Speak "next".
  54    83          Change to M.L.
  55    7E 02 0B    JMP 020B to poll sound (next participant's request for cup).
```

PHONEMES FOR "O.K."

```
0260-68:  26,26,19,06,06,06,06,06; FF (return).
```

PHONEMES FOR WORD "NEXT."

```
0270-09:  0D,02,00,19,19,1F,2A,2A,3E;  FF (return).
```

WELCOMING SPEECH.

```
0280 (welcome):     2D,02,00,18,19,32,23,0C,3E;
0289 (to):          2A,37,37,3E;
028D (robotics):    2B,35,37,0E,15,23,2A,0B,09,19,1F, 3E;
0299 (and):         2F,00,0D,1E,3E;
```

```
029E (Donville's):      1E,15,23,0D,0F,0B,09,18,1F,3E;
02A8 (doughnuts):       1E,26,0D,32,31,2A,1F,3E,3E,3E;
02B2 (ask):             2F,2F,1F,19,3E;
02B7 (me):              0C,3C,29,3E;
02BB (for):             1D,34,34,2B,3E;
02C0 (a):               06,21,29,3E;
02C4 (cup):             19,32,31,25,3E;
02C9 (for):             1D,34,34,2B,3E;
02CE (coffee):          19,3D,1D,3C,29,3E,3E,3E,3E,3E,3E;
02D9 (Dave):            1E,06,09,29,0F,3E;
02DF (serves):          1F,3A,3A,0F,12,3E;
02E5 (doughnuts):       1E,26,0D,32,31,2A,1F,3E,3E,3E; FF (return).
02F0 (I repeat):        15,00,09,29,3E; 2B,3C,25,3C,3C,2A,3E,3E,3E; FF (return).

                        PHONEMES FOR "PRESS KEY, TAKE CUP."

0300 (press):           25,2B,02,1F,3E;
0305 (key):             19,2C,2C,3E,3E,3E;
030B (take):            2A,06,21,29,29,3E;
0311 (cup):             19,32,31,25,3E,3E,3E; FF (return).
```

Run the program and do exactly what is stated in the formulation section. Observe carefully the actions and events.

PROBLEM

If you're brave, try having the robot serve coffee to the participant after coming back with the cup.

EXPERIMENT 8-11

HERO GOES TO A DANCE.

The author wishes to express thanks to G. W. Gladish and R. A. Rudolf of the Western Electric Co. (Indianapolis) for their permission to include this HERO song and dance experiment, which they conceived and implemented. The program below is entirely theirs. Thanks are also due Western Electric for permission to reproduce the program and the use of their name. Messrs. Gladish and Rudolf conceived, formulated, and solved this problem during a robot course they attended at W.E. July 1983. It's rather amusing. Perhaps "working/dancing robots" are the answer to possible resistance to their use in various areas of society's activities.

We'll let the program and its commentary speak (sing and dance) for itself.

```
0A00    3F              Change to R.L.
  01    FD 00 00        Initialize the robot with wrist pivot at 50 and steer at 00.
        4D,50,00
        62 00
  09    72 FA 4B        Speak the phonemes at FA4B ("hello, my name is HERO").
```

```
0C      72 0C 00    Speak the phonemes at 0C00 ("do you want to dance, honey?")
0F      83          Change to M.L.
10      CE 00 00    LDX 0000.
13      3F          Change to R.L.
14      D3 48 04    Raise arm 4 units (slow).
17      38 38 04    Extend arm 4 units.
1A      83          Change to M.L.
1B      08          INX.  Increment index register.
1C      8C 00 20    CPX 0020.  Compare (X) with 0020.
1F      26 F2       BNE F2.  If (X)≠0020, branch back to 0A13 and keep raising
                    and extending arm 128 dec (80H) times till arm level is at
                    80H.  (4x32=128).
21      3F          Change to R.L.
22      CC F8 00    Steer left (fast), continue.
25      72 0B 00    Speak the phonemes at 0B00 ("go, man").
28      1D FB       Branch if steering busy (back to 0A25 and speak again while
                    steering).
2A      83          Change to M.L.
2B      86 05       LDAA 05 (number of "hand waves").
2D      3F          Change to R.L.
2E      D3 1C 05    Drive fast, reverse, 05 units.
31      D3 90 20    Wrist pivots down 20 units.
34      D3 94 20    Wrist pivots up 20 units.
37      D3 94 20    Wrist pivots up 20 units.
3A      D3 90 20    Wrist pivots down 20 units (0A31 through 0A3C constitute a
                    "hand wave").
3D      83          Change to M.L.
3E      4A          DECA
3F      81 00       CMPA 00.  Is (A)=00?
41      26 EA       BNE EA.  No.  Branch back to 0A2D and wave hand again as body
                    moves back again.
43      3F          Change to R.L.
44      CC D8 A0    Move head to A0, continue.
47      72 0B 00    Speak "go, man".
4A      1E FB       Branch if arm (head) busy back to 0A47.  Speak "go, man"
                    again.
4C      CC D8 60    Move head back to 60.
4F      72 0B 00    "Go, man".
52      1E FB       Branch if head busy to 0A4F.
54      CC D8 30    Move head back to 30.
57      72 0B 00    "Go, man".
5A      1E FB       Branch if head busy to 0A57.
5C      CC D8 60    Move head back to 60.
5F      72 0B 00    "Go, man".
62      1E FB       Branch if head busy to 0A5F.
64      83          Change to M.L.
65      BD F6 4E    JSR REDIS
68      86 60       LDAA 60.
6A      BD F7 AD    JSR OUTBYT.  See 60 (for Go).
6D      BD F7 AD    JSR OUTBYT.  Another Go.
70      BD F7 AD    JSR OUTBYT.  Another Go.  Thus see "Go Go Go" on display.
73      7E 0A 21    JMP 0A21.  Keep on dancing and singing.
```

"DO YOU WANT TO DANCE HONEY?"

```
0C00 (do):          1E,36,37,37,3E;
  05 (you):         22,36,37,37,3E;
  0A (want):        2D,08,0D,2A,3E;
  0F (to):          2A,37,37,3E;
  13 (dance):       1E,2F,0D,1F,1F,3E,3E;
  1A (honey):       1B,32,32,0D,0D,3C,3C,FF (return).
```

"GO MAN"

```
0B00:  1C,35,37,3E; 0C,2F,00,0D,3E,3E; FF (return).
```

Run the program and observe. It should be fun.

EXPERIMENT 8-12

A ROBOT SORTS PARTS COMING DOWN A CONVEYOR BELT ACCORDING TO SIZE AND PLACES THEM IN APPROPRIATE BINS.

The practical industrial applications for this experiment are evident from the title.

FORMULATION

You are to think of a conveyor belt moving parts of (here) 3 different sizes, one behind the other down the belt. The mechanical arrangement is such that size 1, if it passes a certain point just before reaching the robot, trips switch S1; size 2 trips switch S2; and size 3 trips switch S3. We simulate this arrangement with the three switches S3,S2,S1 on HERO's experimental board inport C2A0 (they are input to data bus lines D3,D2,D1 respectively). Depending on which switch is tripped (from lo to hi to lo) the robot will open its gripper and close it the right amount to grab that part (the amount the gripper must close depends on the size of that part). The head then is to turn to a position where a bin is to receive that part. The robot opens and completely closes its gripper releasing the part to the bin. The head then turns back to the conveyor belt to await another part. We shall have bins 1,2,3 (corresponding to parts of sizes 1,2,3) located at head positions 52,42, and 32 respectively. As usual, the neutral position of the head will be 62 (facing the conveyor belt). We'll take the closed gripper positions to be 30,20, and 10 to fit around sizes 3,2,1 respectively. The display is to show 01,02, or 03 during the time size 1,2, or 3 is being serviced.

POLLING PORT C2A0 FOR THE TRIPPED SWITCH (SIZE OF PART).

```
0500     BD F6 5B     JSR CLRDIS.  Clear display.
  03     B6 C2 A0     LDAA C2A0.  Input port C2A0 status.
  06     85 02        BITA 02.  S1 thrown hi (size 1)?
  08     26 16        BNE 16.  Yes.  Branch to 0520 and service size 1 (put into
                      bin 1).
  0A     85 04        BITA 04.  No.  S2 hi (size 2)?
```

```
 0C      26 32        BNE 32.  Yes.  Branch to 0540 and put size 2 into bin 2.
 0E      85 08        BITA 08.  No.  S3 hi (size 3)?
 10      26 4E        BNE 4E.  Yes.  Branch to 0560 and put size 3 into bin 3.
 12      20 EC        BRA EC.  None tripped.  Branch back to 0500 to keep polling
                      for S1,S2,S3.
```

S1 HI. SIZE 1. DISPLAY 01. TAKE PART TO BIN 1 AND GO BACK.

```
0520     BD F6 4E     JSR REDIS
 23      86 01        LDAA 01
 25      BD F7 AD     JSR OUTBYT.  See 01 on display.
 28      3F           Change to R.L.
 29      C3 A8 50     Open gripper to 50 (slow).
 2C      C3 A8 10     Close it to 10 (around size 1).
 2F      C3 D0 52     Rotate head to 52 (bin 1).
 32      C3 A8 20     Open gripper to 20 (release size 1).
 35      C3 A8 00     Close gripper completely.
 38      C3 D0 62     Rotate head back to 62 (conveyor).
 3B      83           Change to M.L.
 3C      7E 05 00     JMP 0500.  Poll for new size.
```

S2 HI. SIZE 2. DISPLAY 02. TAKE PART TO BIN 2 AND GO BACK.

```
0540     BD F6 4E     JSR REDIS
 43      86 02        LDAA 02
 45      BD F7 AD     JSR OUTBYT.  Display 02.
 48      3F           Change to R.L.
 49      C3 A8 50     Open gripper to 50.
 4C      C3 A8 20     Close it to 20 for size 2.
 4F      C3 D0 42     Rotate head to 42 (bin 2).
 52      C3 A8 30     Open gripper to 30 (release size?)
 55      C3 A8 00     Close gripper completely.
 58      C3 D0 62     Rotate head back to 62 (conveyor).
 5B      83           Change to M.L.
 5C      7E 05 00     JMP 0500.  Poll for new size.
```

S3 HI. SIZE 3. DISPLAY 03. TAKE PART TO BIN 3 AND GO BACK.

```
0560     BD F6 4E     JSR REDIS
 63      86 03        LDAA 03.
 65      BD F7 AD     JSR OUTBYT.  Display 03
 68      3F           Change to R.L.
 69      C3 A8 50     Open gripper to 50.
 6C      C3 A8 30     Close it to 30 to take size 3.
 6F      C3 D0 32     Rotate head to 32 (bin 3).
 72      C3 A8 40     Open gripper to 40 (release size 3).
 75      C3 A8 00     Close gripper completely.
 78      C3 D0 62     Rotate head back to 62 (conveyor).
 7B      83           Change to M.L.
 7C      7E 05 00     JMP 0500.  Poll for new size.
```

PROCEDURE

Run the program. Throw any one of switches S1,S2,S3 hi (then back lo). See 01,02, or 03 on the dipslay depending on whether switch S1,S2, or S3 was thrown (size 1,2, or 3). Observe the gripper open and close to a position commensurate with size 1,2, or 3 (gripper positions 10,20,30 respectively) followed by the head rotating to position 52,42 or 32 (bin 1,bin2, or bin 3 for sizes 1,2,3 respectively). The gripper then opens a bit (to 20,30, or 40 to release size 1,2, or 3 into the bin). It then closes fully, the head rotates back to 62 (conveyor), and the display clears. Throw another switch lo-hi-lo. The sequence of events will repeat appropriate to the size of the part detected and its bin. Do this several times throwing S1,S2, or S3 lo-hi-lo in any order you please.

PROBLEM

Modify the program in the appropriate places so that each size part is "treated" or "worked on" in a manner specified for that part before being placed in the right bin. Think up any and all other variations that would have relevance to an industrial manufacturing setting e.g. painting different parts with different colors; or, instead of bins, placing each part in its proper slot in a stack of slots (sensing by light whether the slot is empty or not). This is a very important problem.

EXPERIMENT 8-13

HERO USED IN X-Y PLOTTING (APPLICATIONS TO GRAPHICS, MACHINE PATTERN CUTTING, SPRAY PAINTING, NUMERIC CONTROLS, ETC).

PART A: RECTANGLES.

FORMULATION

1. After a "31" initialization, the program is executed and the gripper opens slightly, the arm extends, then the gripper closes (around a pencil or, preferably, a magic marker held horizontally).
2. The display blanks.
3. User is to key in the dimensions X,Y where X will be distance the robot travels first in the forward direction and later back in reverse direction (X axis motion) while Y will represent the vertical distance the arm will retract and later extend (Y axis motion). XY shows on the display (X,Y = 1,2,..., or F).
4. Press any key to initiate plotting of the rectangle. Hold a pad firmly and stationary next to the marker so it can trace out the rectangle on the paper.
5. When the rectangle is completed, a time delay of 3 or 4 seconds is entered after which another rectangle of the same size is plotted unless during this time interval you throw switch S3 at port C2A0 on the experimental board hi, in which case you go back to (2) above to key in new X,Y rectangle dimensions for a new sized rectangle. All then proceeds as before.
6. In what follows, consult the table in Appendix B.

INITIALIZATIONS

```
0100    3F          Change to R.L.
  01    C3 A8 10    Open gripper slowly to 10H.
  04    C3 28 20    Extend arm slowly to 20H.
  07    C3 A8 00    Close gripper (around marker).
  0A    83          Change to M.L.
  0B    BD F6 5B    CLRDIS.  Blank display.
  0E    BD F6 4E    REDIS.
  11    BD F7 96    JSR IHB.  Input two keys X,Y.  See XY on display.  XY is in
                    ACCA.  X,Y = width, height of rectangle.
  14    16          TAB.  Save XY in ACCB.
  15    BD F7 77    JSR INCH.  Press any key to proceed with plot.
```

PLOT RECTANGLE

```
0118    17          TBA.  XY to ACCA.
  19    84 0F       ANDA (with 0F).  0Y in ACCA.
  1B    3F          Change to R.L.
  1C    D3 2C 01    Retract arm 01 units slowly (in Y direction).
  1F    83          Change to M.L.
  20    4A          DECA.
  21    26 F8       BNE F8 back to 011B.  Y motion is not complete.
  23    17          TBA.  XY to ACCA.  Y motion done.
  24    44          LSRA.
  25    44
  26    44
  27    44          0X is now in ACCA.
  28    3F          Change to R.L.
  29    D3 08 01    Move forward 01 units slowly in X direction.
  2C    83          Change to M.L.
  2D    4A          DECA.
  2E    26 F8       BNE F8 to 0128.  X motion not complete.
  30    17          TBA.  XY to ACCA.
→ 31    84 0F       ANDA.  0Y in ACCA.
  33    3F
  34    D3 28 01    Extend arm 1 unit slowly.
  37    83
  38    4A
  39    26 F8       BNE F8 to 0133.  Y motion not complete..
  3B    17          TBA.  XY to ACCA.
  3C    44
  3D    44
  3E    44
  3F    44          0X now in ACCA.
  40    3F
  41    D3 0C 01    Drive in reverse 01 units, slowly.
  44    83
  45    4A
  46    26 F8       BNE back to 0140.
```

RECTANGLE COMPLETE. ENTER TIME AND POLL SWITCH S3.

```
0148    37          PSHB.  Save ACCB on stack.
```

```
 49      C6 04          LDAB 04 for time (about 4 sec.).
 4B      CE FF FF       LDX FFFF for 4 sec. time.
 4E      B6 C2 A0       LDAA C2A0.  Poll S3 at port C2A0.
 51      85 08          BITA 08.  Test S3 for hi or lo.
 53      26 B6          BNE to 010B.  S3 was hi.  Blank display and key in new X,Y
                        dimensions for a new sized rectangle.
 55      09             DEX.  S3 not hi.  Continue time and polling of S3.
 56      26 F6          BNE to 014E to poll S3.
 58      5A             DECB.
 59      26 F0          BNE to 014B.  Time not over (B not 0).

          TIME OVER.  S3 NOT HI.  DO ANOTHER RECTANGLE OF SAME SIZE.

015B     33             PULB.  Restore B from stack.
  5C     7E 01 18       JMP to 0118 to do another rectangle.
```

Execute the above program and proceed as indicated in the formulation above.

PART B: "SINE CURVES".

FORMULATION

1. After a "31" initialization, the program is executed. The gripper opens, arm extends, and gripper closes (around pencil or marker to be held horizontally).
2. Press any key to initiate a sine curve.
3. The robot will move forward as the arm retracts, then extends, and then retracts. Robot stops forward motion precisely when the last retraction is complete. The marker will plot one cycle of an approximate sine curve.
4. If you now press any key the robot will go back and then extend its arm to get into position to do another sine curve.
5. Pressing any key will initiate another sine curve.

```
0200     C3 A8 10       Open gripper slowly to 10H.
  03     3F             Change to R.L.
  04     C3 28 20       Extend arm to 20H slowly.
  07     C3 A8 00       Close gripper (around a marker).
  0A     83             Change to M.L.
  0B     BD F7 77       JSR INCH.  Wait.  Press key to initiate plotting.
  0E     86 41          LDAA 41.  41 to ACCA
  10     B7 C2 A0       STAA C2A0.  41 to port C2A0 (main drive circuit board).  41
                        will produce forward motion at very slow speed (see Expt.
                        2-12).  Robot moves forward.
  13     3F
  14     D3 3C 10       Retract arm fast 10H units.
  17     D3 38 10       Extend arm fast 10H units.
  1A     D3 3C 10       Retract arm fast 10H units.
  1D     83             Change to M.L.
  1E     86 01          LDAA 01.
  20     B7 C2 A0       01 to port C2A0 will stop forward motion (see Expt. 2-12).
                        One sine curve done.
  23     BD F7 77       JSR INCH.  Press any key to send robot back.
  26     3F             Change to R.L.
  27     D3 0C 10       Drive robot back 10H units.
```

2A	02	Abort drive.
2B	83	Change to M.L.
2C	7E 02 03.	JMP 0203 to extend arm and prepare to execute another sine curve.

Follow the steps outlined in the formulation.

PROBLEM

Follow our method or any other of your own to have HERO plot all kinds of other curves. Use any of the 7 degrees of freedom to implement your method. Extend to 3 dimensions.

You can surely see the many applications that this experiment would open up in the areas of machine control, pattern cutting, tooling, numeric controls, graphics, spray painting etc.

This experiment, like others throughout the book, should point out to you that successful robot implementation of an application often depends, in a very crucial way, on correct parts presentation, proper gripping procedures and careful follow-through. These "side-issues" are as important as the purely robot considerations.

EXPERIMENT 8-14

LIMIT SWITCH CONTROL of Different Activities In AN ASSEMBLY PROCESS

In This expt. we shall have switch S0, S1, or S2 at Port C2A0 on the experimental board choose a specific robot activity, depending on which switch is thrown hi. If all are low there is to be no robot activity. If one is thrown hi, a specific activity is chosen and continues till either it is thrown back lo, when activity ceases, or another is thrown hi, choosing another activity. You may think of the switches as limit switches situated at different positions on an assembly line. When the "widgit" reaches a new assembly line position (tripping the switch there) a different manufacturing activity (operation) is called for in the assembly process. When the switch is released after the widgit passes, activity ceases till the widgit reaches the next position on the line tripping the switch there to initiate the next activity in the process. Of course, the switch may also represent more generally a sensor/detector of any kind (infra-red, light, vision etc). The program (next page) would be the same except for the address of the port inputting the sensors status. Thus:

$\frac{/S0}{\text{Position A}}$ $\frac{/S1}{\text{Pos. B}}$ $\frac{/S2}{\text{Pos C}}$ where S0, S1, S2 can be switches, sensors, etc (The sensors outputting 1s or 0s),

← Assembly Line Positions →

Each activity finishes up before the switches are polled again

```
0500  B6 C2 A0   Input S1 from Exptl. Board
  03  85 01      S0 hi?
  05  26 0B      yes, BNE 0512 to move arm up, then down
  07  85 02      NO, S1 hi?
  09  26 12      Yes BNE 051D to open, then close gripper
  0B  85 04      NO. S2 hi?
  0D  26 19      Yes, BNE 0528 to move head CCW, then back.
  0F  7E 05 00   JMP 0500. Poll switches again
```

Activity #1: S0 hi

```
0512  3F
  13  C3 50 20 ] send arm up, then down
  16  C3 50 00 ]
  19  83
  1A  7E 05 00  Poll S1 again
```

Activity #2: S1 hi

```
051D  3F
  1E  C3 B0 30 ] open, then close, the gripper
  21  C3 B0 00 ]
  24  83
  25  7E 05 00  Poll S1 again
```

Activity #3: S2 hi

```
0528  3F
  29  C3 D0 52 ] move head CCW, then back
  2C  C3 D0 62 ]
  2F  83
  30  7E 05 00  Poll S1 again
```

Each activity finishes its "job" before the switches are polled again for another (or same) activity.

APPENDIX A

USEFUL HERO REFERENCE MATERIAL ON PROGRAMMING AND I/Os. THE 6800 INSTRUCTION SET. 6808 PIN-OUT.

The material in this appendix pertinent to the Heath HERO is reproduced with the kind permission of the Heath Co. (subsidiary of Zenith Radio Inc.). The 6800/6808 microprocessor instruction set and the 6808 pin-out are reproduced courtesy of Motorola Inc.

From Heath Robot Technical Manual.

PROGRAMMER'S INFORMATION SHEET

Robot owners may copy this sheet to use as a handy aid for use when programming their Robot.

INTERPRETER COMMANDS

COMMAND FORM*	TITLE
02	Abort Drive Motor
03	Abort Steering Motor
04	Abort Arm Motors †
05	Abort Speech
1C OO	Branch if Base Busy
1D OO	Branch if Steering Busy
1E OO	Branch if Arm Busy †
1F OO	Branch if Speech Busy
21	Zero
3A	Return to Executive ("Ready")
(3F)	(Change to Robot Language)
41	Enable Light Detector
42	Enable Sound Detector
45	Enable Ultrasonic Ranging
4B	Enable Motion Detector
4E	Enable Display
51	Disable Light Detector
52	Disable Sound Detector
55	Disable Motion Detector
5B	Disable Ultrasonic Ranging
5E	Disable Display
61 OO	Speak, Continue (index)
62 OO	Speak, Wait (index)
71 MM MM	Speak, Continue (extended)
72 MM MM	Speak, Wait (extended)
83	Change to Machine Language [3F, SW] changes back to Robot]
87 XX XX	Sleep (immediate)
8F XX XX	Pause (immediate)
BF MM MM	Jump if Speaking (extended)
C3 SS XX	Motor Move, wait abs (immediate)
CC SS XX	Motor Move, continue abs (immed)
D3 SS XX	Motor Move, wait rel (immediate)
DC SS XX	Motor Move, continue rel (immed)
E3 OO	Motor Move, wait abs (index)
EC OO	Motor Move, continue abs (index)
F3 MM MM	Motor Move, wait abs (extended)
FC MM MM	Motor Move, continue abs (extended)
FD**	Motors, Move All abs (immediate)

* XX = distance, position, time, etc. byte
SS = select motor, speed, direction byte
MM = memory address byte
OO = offset number byte

**Seven motor position bytes: extend, shoulder, rotate, pivot, gripper, head, and steering.

† Arm here refers to arm, extend, rotate, pivot, gripper, head.

MOTOR CONTROL COMMANDS (Byte 1 = Opcode)

Byte 2	Byte 3
hex no. = S S	hex no. = X X
binary = mmms sDdd	binary = dddd dddd

where:

mmm are the motor select bits.
- 000 = drive motor.
- 001 = extend motor.
- 010 = shoulder motor.
- 011 = wrist rotate motor.
- 100 = wrist pivot motor.
- 101 = gripper motor.
- 110 = head motor.
- 111 = steering motor.

ss are the speed select bits:
- 01 = slow.
- 10 = medium.
- 11 = fast.

D selects which way the motors run (for relative commands only). If D equals:
- 1 = the position increases as the motor runs.*
- 0 = the position decreases as the motor runs.*

*"Position" is the number stored in memory for that motor.

dd dddd dddd = "Position" — how far, or to where, the motor turns. Highest 2 bits are for drive motor only.

SUBROUTINES

INCH (F777)	Key debounce.
OUTCH (F7C8)	Create character.
OUTHEX (F7B5)	Output 1 hex character.
OUTBYT (F7AD)	Output 2 hex characters.
CLRDIS (F65B)	Clear display, set DIGADD.
OUTSTR (F7E5)	Output character string.
DSPLAY (F6F9)	OUTBYT, used 1, 2, or 3 times.
IHB (F796)	Input hex byte, to display and accumulator A.
REDIS (F64E)	Reset DIGADD for left-most position.
DIGADD (0FF2, 0FF3)	Location for storing the next character's display position.

MEMORY ADDRESSES AND RANGES

(Useable RAM 003F to 0EE0)

Motor	Address	Range
Extend Position	0000	(in)00–98(out)
Shoulder Position	0001	(dn)00–86(up)
Rotate Position	0002	(CCW)00–93(CW)*
Pivot Position	0003	(up)00–A5(dn)
Gripper Position	0004	(closed)00–75(open)
Head Position	0005	(CCW)00–C2(CW)
Steering	0006	(L turn)00–93(R turn)
Ultrasonic Ranging Hits	0010	
Ultrasonic Range	0011	

READING CLOCK DATA

1. STAA (A0) at $C300.
2. STAA (variable select) at $C300. C2C Ø.
3. LDAA (time variable) from $C300.

INPUT (VARIABLE SELECT)	OUTPUT (TIME VARIABLE)	
00	SEC(1)	
01	SEC(10)	
02	MIN(1)	
03	MIN(10)	
04	HR(1)	
05	HR(10)	+4 = P.M. +8 = 24 HR. CLOCK
06	WEEK OF THE MONTH	
07	DAY(1)	
08	DAY(10)	+4 = FEB. LEAP YEAR
09	MO(1)	
0A	MO(10)	
0B	YR(1)	
0C	YR(10)	

READING 1024 HZ COUNT-TO-$FF (256)

1. CLI
2. STAA (AF) at C300
3. LDAA from 0EFC

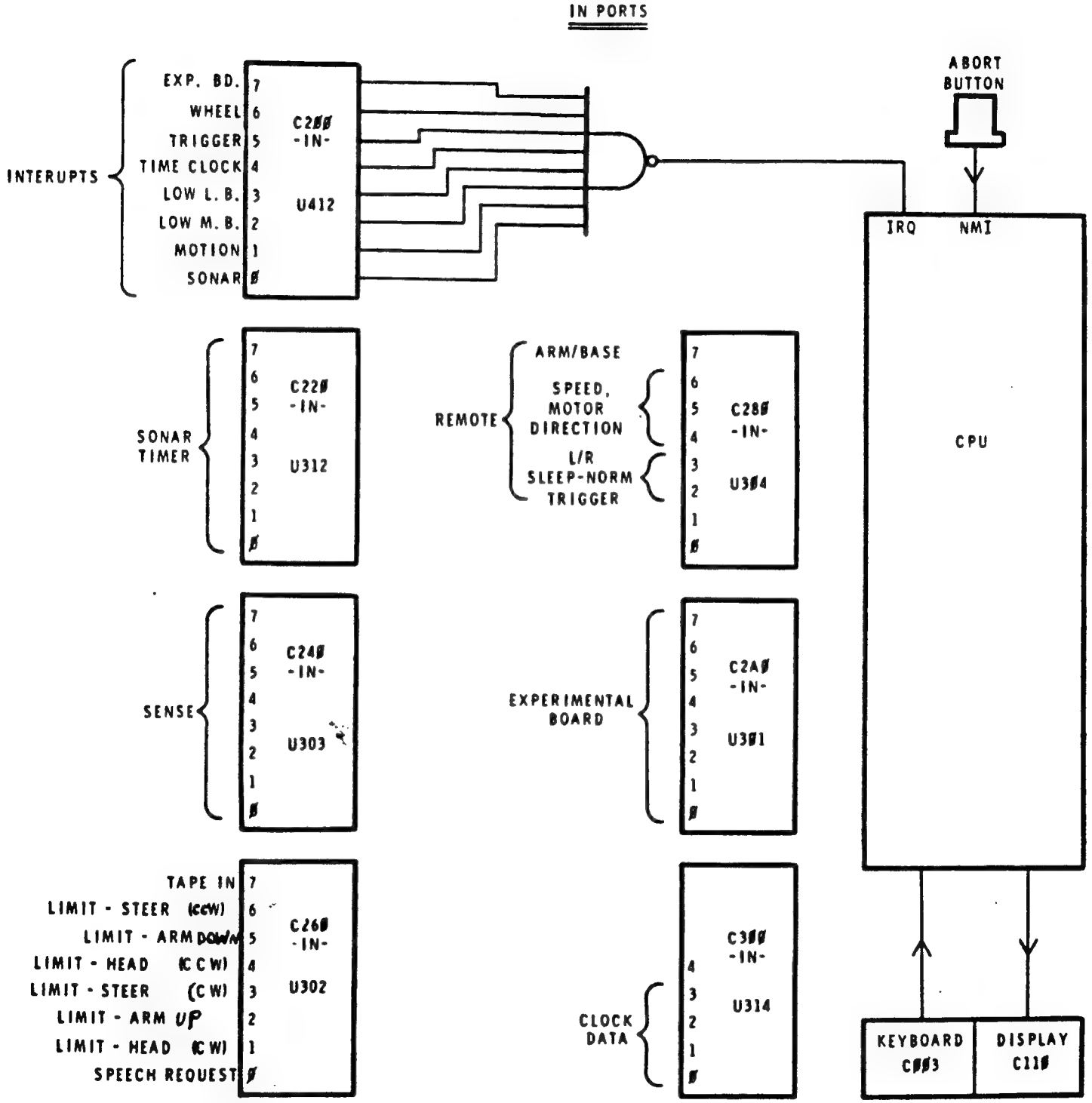
AUTOMATIC JUMPS

(Executive Mode)	
User 1	0030, 0031, 0032.
User 2	0033, 0034, 0035.
User 3	0036, 0037, 0038.

INTERRUPT VECTORS

Motion Detect	0027, 0028, 0029.
Trigger	002A, 002B, 002C.
Experimental Board IRQ (low)	002D, 002E, 002F.

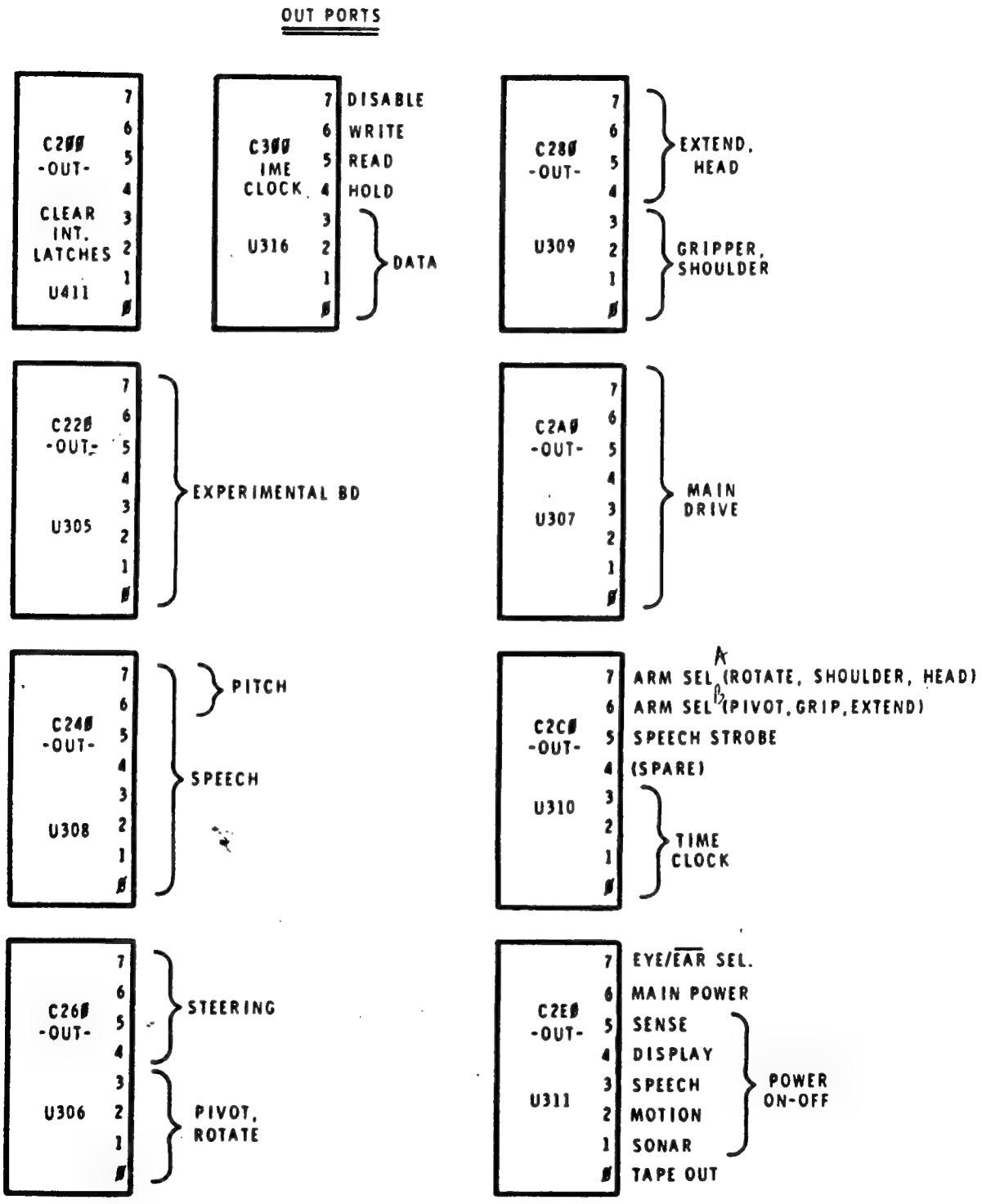
From Heath Robot Technical Manual.



PORT MAP

APPENDIX A

From Heath Robot Technical Manual.



From Heath Robot Technical Manual.

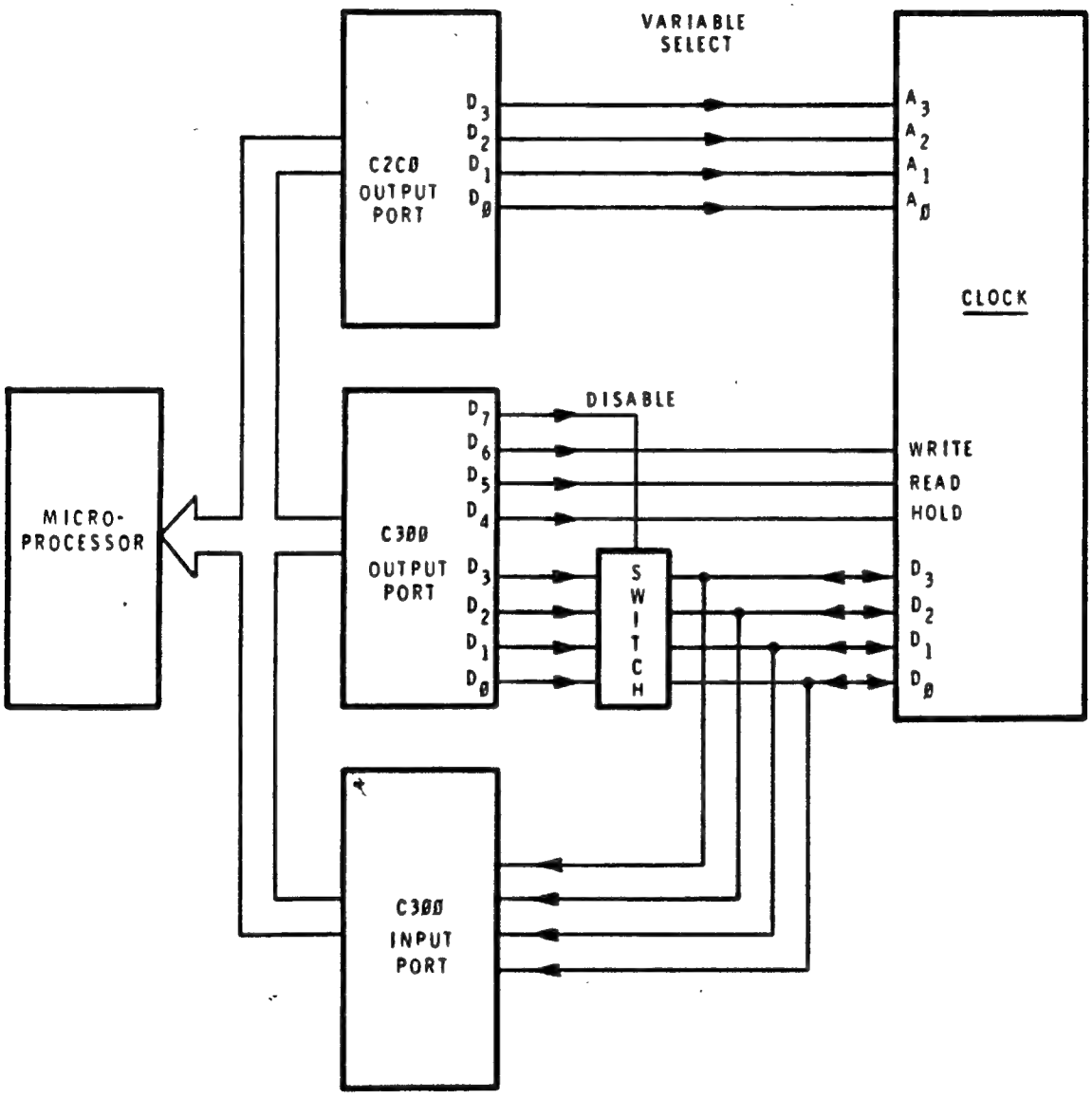


FIGURE 2

HERO's Real Time Clock.

From Heath course notes "Robotics and Industrial Electronics"

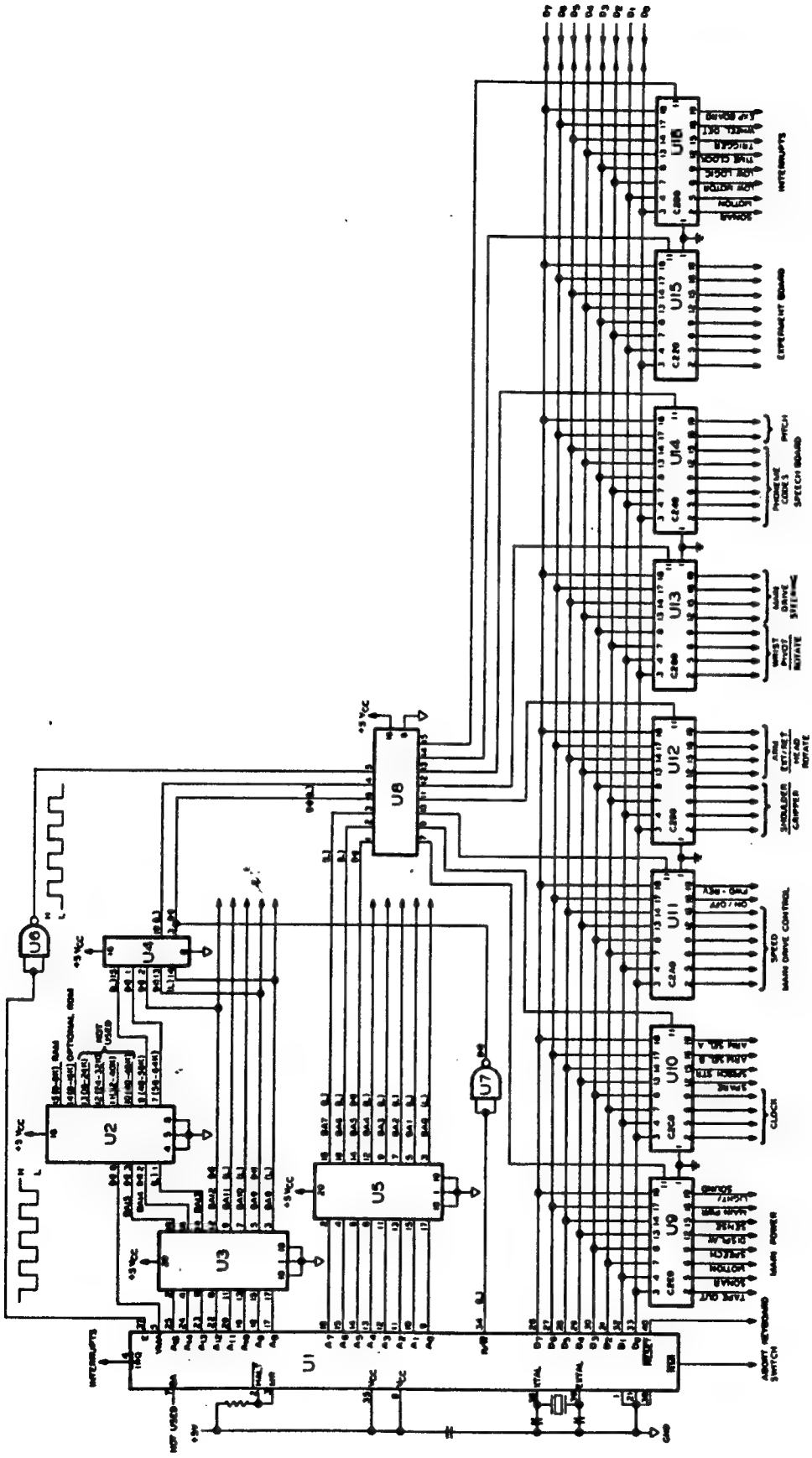


Figure 10-26
ET-18 CPU and I/O circuitry for outputting data from the MPU.

U9-U16 are octal D flip flops:74374 latches.
Pin 11 (clock) must go low to high to latch in.

From Heath course notes "Robotics and Industrial Electronics".

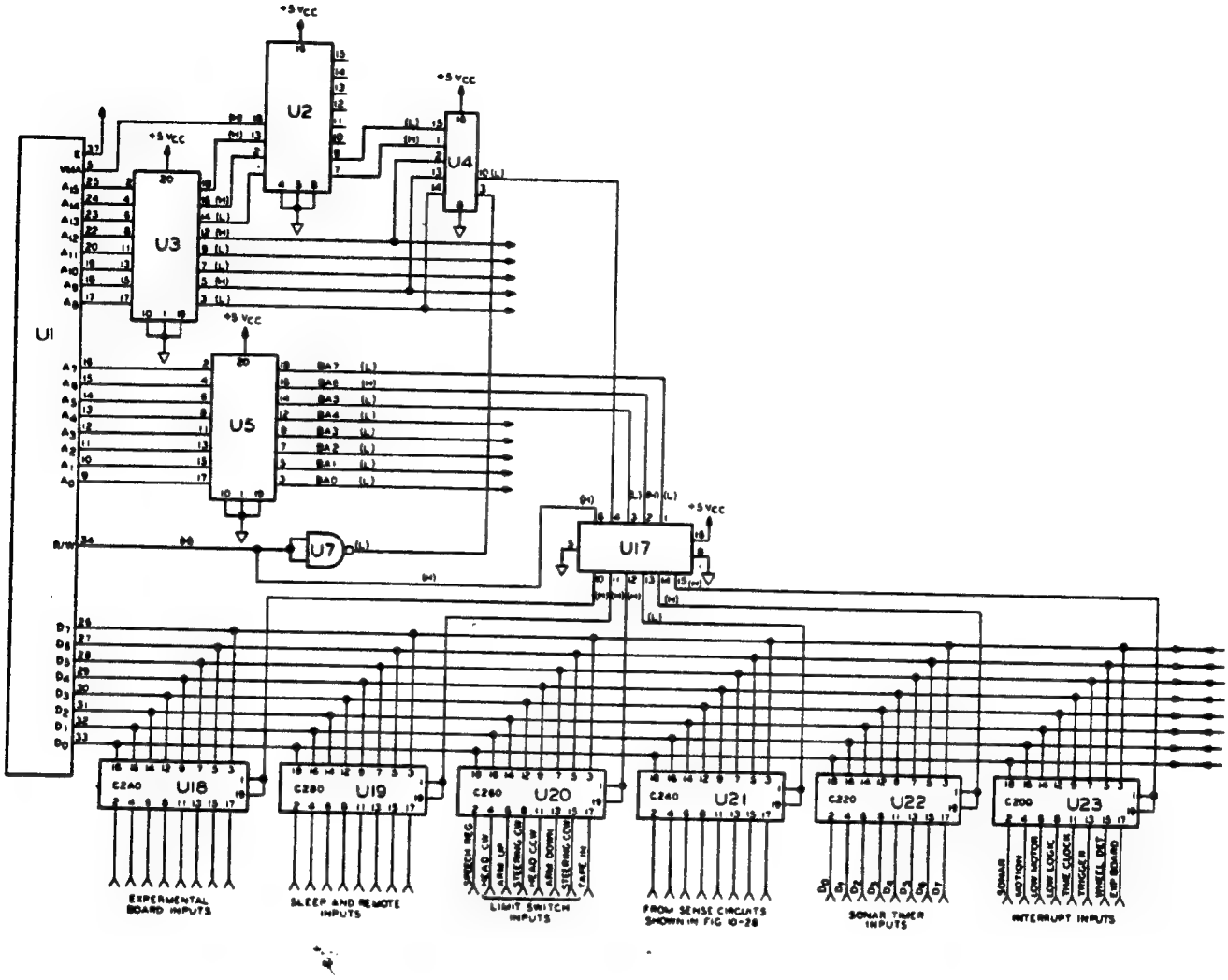


Figure 10-29
ET-18 CPU and I/O circuitry for inputting data to the MPU.

U18-U23 are octal tri-state buffers:74244.
Pins 1 and 19 must go low to de-tristate.

TABLE 3 – ACCUMULATOR AND MEMORY INSTRUCTIONS

OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED =	DIRECT OP	DIRECT ~	DIRECT =	INDEX OP	INDEX ~	INDEX =	EXTND OP	EXTND ~	EXTND =	IMPLIED OP	IMPLIED ~	IMPLIED =	BOOLEAN/ARITHMETIC OPERATION (All register labels refer to contents)	5 H	4 I	3 N	2 Z	1 V	0 C
Add	ADDA	8B	2	2	9B	3	2	AB	5	2	BB	4	3				A + M → A	↕	•	↕	↕	↕	↕
	ADDB	CB	2	2	DB	3	2	EB	5	2	FB	4	3				B + M → B	↕	•	↕	↕	↕	↕
Add Acmltrs	ABA													1B	2	1	A + B → A	↕	•	↕	↕	↕	↕
Add with Carry	ADCA	89	2	2	99	3	2	A9	5	2	B9	4	3				A + M + C → A	↕	•	↕	↕	↕	↕
	ADCB	C9	2	2	D9	3	2	E9	5	2	F9	4	3				B + M + C → B	↕	•	↕	↕	↕	↕
And	ANDA	84	2	2	94	3	2	A4	5	2	B4	4	3				A · M → A	•	•	↕	↕	R	•
	ANDB	C4	2	2	D4	3	2	E4	5	2	F4	4	3				B · M → B	•	•	↕	↕	R	•
Bit Test	BITA	85	2	2	95	3	2	A5	5	2	B5	4	3				A · M	•	•	↕	↕	R	•
	BITB	C5	2	2	D5	3	2	E5	5	2	F5	4	3				B · M	•	•	↕	↕	R	•
Clear	CLR							6F	7	2	7F	6	3				00 → M	•	•	R	S	R	R
	CLRA													4F	2	1	00 → A	•	•	R	S	R	R
	CLRB													5F	2	1	00 → B	•	•	R	S	R	R
Compare	CMPA	81	2	2	91	3	2	A1	5	2	B1	4	3				A - M	•	•	↕	↕	↕	↕
	CMPB	C1	2	2	D1	3	2	E1	5	2	F1	4	3				B - M	•	•	↕	↕	↕	↕
Compare Acmltrs	CBA													11	2	1	A - B	•	•	↕	↕	↕	↕
Complement, 1's	COM							63	7	2	73	6	3				M → M	•	•	↕	↕	R	S
	COMA													43	2	1	A → A	•	•	↕	↕	R	S
	COMB													53	2	1	B → B	•	•	↕	↕	R	S
Complement, 2's (Negate)	NEG							60	7	2	70	6	3				00 - M → M	•	•	↕	↕	①	②
	NEGA													40	2	1	00 - A → A	•	•	↕	↕	①	②
	NEGB													50	2	1	00 - B → B	•	•	↕	↕	①	②
Decimal Adjust, A	DAA													19	2	1	Converts Binary Add of BCD Characters into BCD Format	•	•	↕	↕	↕	③
Decrement	DEC							6A	7	2	7A	6	3				M - 1 → M	•	•	↕	↕	④	•
	DECA													4A	2	1	A - 1 → A	•	•	↕	↕	④	•
	DECB													5A	2	1	B - 1 → B	•	•	↕	↕	④	•
Exclusive OR	EORA	88	2	2	98	3	2	A8	5	2	B8	4	3				A⊕M → A	•	•	↕	↕	R	•
	EORB	C8	2	2	D8	3	2	E8	5	2	F8	4	3				B⊕M → B	•	•	↕	↕	R	•
Increment	INC							6C	7	2	7C	6	3				M + 1 → M	•	•	↕	↕	⑤	•
	INCA													4C	2	1	A + 1 → A	•	•	↕	↕	⑤	•
	INCB													5C	2	1	B + 1 → B	•	•	↕	↕	⑤	•
Load Acmltr	LDAA	86	2	2	96	3	2	A6	5	2	B6	4	3				M → A	•	•	↕	↕	R	•
	LDAB	C6	2	2	D6	3	2	E6	5	2	F6	4	3				M → B	•	•	↕	↕	R	•
Or, Inclusive	ORAA	8A	2	2	9A	3	2	AA	5	2	BA	4	3				A + M → A	•	•	↕	↕	R	•
	ORAB	CA	2	2	DA	3	2	EA	5	2	FA	4	3				B + M → B	•	•	↕	↕	R	•
Push Data	PSHA													36	4	1	A → M_{SP}, SP - 1 → SP	•	•	•	•	•	•
	PSHB													37	4	1	B → M_{SP}, SP - 1 → SP	•	•	•	•	•	•
Pull Data	PULA													32	4	1	SP + 1 → SP, M_{SP} → A	•	•	•	•	•	•
	PULB													33	4	1	SP + 1 → SP, M_{SP} → B	•	•	•	•	•	•
Rotate Left	ROL							69	7	2	79	6	3				M	•	•	↕	↕	⑤	↕
	ROLA													49	2	1	A	•	•	↕	↕	⑤	↕
	ROLB													59	2	1	B	•	•	↕	↕	⑤	↕
Rotate Right	ROR							66	7	2	76	6	3				M	•	•	↕	↕	⑤	↕
	RORA													46	2	1	A	•	•	↕	↕	⑤	↕
	RORB													56	2	1	B	•	•	↕	↕	⑤	↕
Shift Left, Arithmetic	ASL							68	7	2	78	6	3				M	•	•	↕	↕	⑤	↕
	ASLA													48	2	1	A	•	•	↕	↕	⑤	↕
	ASLB													58	2	1	B	•	•	↕	↕	⑤	↕
Shift Right, Arithmetic	ASR							67	7	2	77	6	3				M	•	•	↕	↕	⑤	↕
	ASRA													47	2	1	A	•	•	↕	↕	⑤	↕
	ASRB													57	2	1	B	•	•	↕	↕	⑤	↕
Shift Right, Logic	LSR							64	7	2	74	6	3				M	•	•	R	↕	⑤	↕
	LSRA													44	2	1	A	•	•	R	↕	⑤	↕
	LSRB													54	2	1	B	•	•	R	↕	⑤	↕
Store Acmltr	STAA				97	4	2	A7	6	2	B7	5	3				A → M	•	•	↕	↕	R	•
	STAB				D7	4	2	E7	6	2	F7	5	3				B → M	•	•	↕	↕	R	•
Subtract	SUBA	80	2	2	90	3	2	A0	5	2	B0	4	3				A - M → A	•	•	↕	↕	↕	↕
	SUBB	C0	2	2	D0	3	2	E0	5	2	F0	4	3				B - M → B	•	•	↕	↕	↕	↕
Subtract Acmltrs	SBA													10	2	1	A - B → A	•	•	↕	↕	↕	↕
Subtr. with Carry	SBCA	82	2	2	92	3	2	A2	5	2	B2	4	3				A - M - C → A	•	•	↕	↕	↕	↕
	SBCB	C2	2	2	D2	3	2	E2	5	2	F2	4	3				B - M - C → B	•	•	↕	↕	↕	↕
Transfer Acmltrs	TAB													16	2	1	A → B	•	•	↕	↕	0	•
	TBA													17	2	1	B → A	•	•	↕	↕	0	•
Test, Zero or Minus	TST							6D	7	2	7D	6	3				M - 00	•	•	↕	↕	R	R
	TSTA													4D	2	1	A - 00	•	•	↕	↕	R	R
	TSTB													5D	2	1	B - 00	•	•	↕	↕	R	R
																		H	I	N	Z	V	C

ADDRESSING MODES — COND. CODE REG.

LEGEND

- OP Operation Code (Hexadecimal).
- ~ Number of MPU Cycles.
- = Number of Program Bytes.
- + Arithmetic Plus.
- - Arithmetic Minus.
- · Boolean AND.
- M_{SP} Contents of memory location pointed to be Stack Pointer.
- + Boolean Inclusive OR.
- ⊕ Boolean Exclusive OR.
- M Complement of M.
- → Transfer into.
- 0 Bit = Zero.
- 00 Byte = Zero.

Note – Accumulator addressing mode instructions are included in the column for IMPLIED addressing.

CONDITION CODE SYMBOLS

- H Half-carry from bit 3.
- I Interrupt mask.
- N Negative (sign bit).
- Z Zero (byte).
- V Overflow, 2's complement.
- C Carry from bit 7.
- R Reset Always.
- S Set Always.
- ↕ Test and set if true, cleared otherwise.
- • Not Affected.

TABLE 4 – INDEX REGISTER AND STACK MANIPULATION INSTRUCTIONS

POINTER OPERATIONS	MNEMONIC	IMMED OP	IMMED ~	IMMED #	DIRECT OP	DIRECT ~	DIRECT #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BOOLEAN/ARITHMETIC OPERATION	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Compare Index Reg	CPX	8C	3	3	9C	4	2	AC	6	2	BC	5	3				$X_H - M, X_L - (M+1)$	•	•	⑦	↕	⑧	•
Decrement Index Reg	DEX													09	4	1	$X - 1 \rightarrow X$	•	•	•	↕	•	•
Decrement Stack Pntr	DES													34	4	1	$SP - 1 \rightarrow SP$	•	•	•	•	•	•
Increment Index Reg	INX													08	4	1	$X + 1 \rightarrow X$	•	•	•	↕	•	•
Increment Stack Pntr	INS													31	4	1	$SP + 1 \rightarrow SP$	•	•	•	•	•	•
Load Index Reg	LDX	CE	3	3	DE	4	2	EE	6	2	FE	5	3				$M \rightarrow X_H, (M+1) \rightarrow X_L$	•	•	⑨	↕	R	•
Load Stack Pntr	LDS	8E	3	3	9E	4	2	AE	6	2	BE	5	3				$M \rightarrow SP_H, (M+1) \rightarrow SP_L$	•	•	⑨	↕	R	•
Store Index Reg	STX				DF	5	2	EF	7	2	FF	6	3				$X_H \rightarrow M, X_L \rightarrow (M+1)$	•	•	⑨	↕	R	•
Store Stack Pntr	STS				9F	5	2	AF	7	2	BF	6	3				$SP_H \rightarrow M, SP_L \rightarrow (M+1)$	•	•	⑨	↕	R	•
Indx Reg → Stack Pntr	TXS													35	4	1	$X - 1 \rightarrow SP$	•	•	•	•	•	•
Stack Pntr → Indx Reg	TSX													30	4	1	$SP + 1 \rightarrow X$	•	•	•	•	•	•

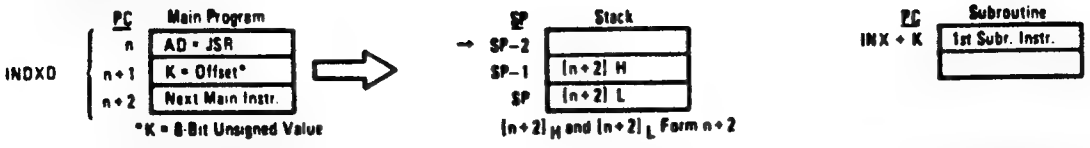
TABLE 5 – JUMP AND BRANCH INSTRUCTIONS

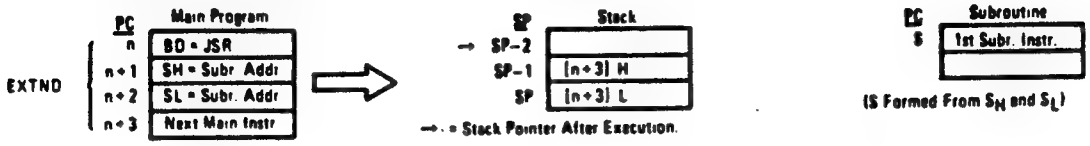
OPERATIONS	MNEMONIC	RELATIVE OP	RELATIVE ~	RELATIVE #	INDEX OP	INDEX ~	INDEX #	EXTND OP	EXTND ~	EXTND #	IMPLIED OP	IMPLIED ~	IMPLIED #	BRANCH TEST	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Branch Always	BRA	20	4	2										None	•	•	•	•	•	•
Branch If Carry Clear	BCC	24	4	2										C = 0	•	•	•	•	•	•
Branch If Carry Set	BCS	25	4	2										C = 1	•	•	•	•	•	•
Branch If = Zero	BEQ	27	4	2										Z = 1	•	•	•	•	•	•
Branch If ≥ Zero	BGE	2C	4	2										$N \oplus V = 0$	•	•	•	•	•	•
Branch If > Zero	BGT	2E	4	2										$Z + (N \oplus V) = 0$	•	•	•	•	•	•
Branch If Higher	BHI	22	4	2										$C + Z = 0$	•	•	•	•	•	•
Branch If ≤ Zero	BLE	2F	4	2										$Z + (N \oplus V) = 1$	•	•	•	•	•	•
Branch If Lower Or Same	BLS	23	4	2										$C + Z = 1$	•	•	•	•	•	•
Branch If < Zero	BLT	2D	4	2										$N \oplus V = 1$	•	•	•	•	•	•
Branch If Minus	BMI	2B	4	2										N = 1	•	•	•	•	•	•
Branch If Not Equal Zero	BNE	26	4	2										Z = 0	•	•	•	•	•	•
Branch If Overflow Clear	BVC	28	4	2										V = 0	•	•	•	•	•	•
Branch If Overflow Set	BVS	29	4	2										V = 1	•	•	•	•	•	•
Branch If Plus	BPL	2A	4	2										N = 0	•	•	•	•	•	•
Branch To Subroutine	BSR	8D	8	2											•	•	•	•	•	•
Jump	JMP				6E	4	2	7E	3	3				See Special Operations	•	•	•	•	•	•
Jump To Subroutine	JSR				AD	8	2	BD	9	3					•	•	•	•	•	•
No Operation	NOP										01	2	1	Advances Prog Cntr. Only	•	•	•	•	•	•
Return From Interrupt	RTI										3B	10	1		———			⑩	———	
Return From Subroutine	RTS										39	5	1		•	•	•	•	•	•
Software Interrupt	SWI										3F	12	1	See Special Operations	•	•	•	•	•	•
Wait for Interrupt*	WAI										3E	9	1		•	⑪	•	•	•	•

*WAI puts Address Bus, R/W, and Data Bus in the three-state mode while VMA is held low.

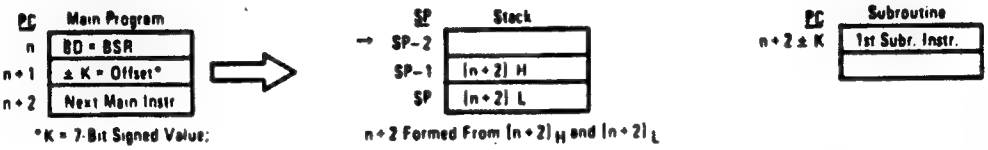
SPECIAL OPERATIONS

JSR, JUMP TO SUBROUTINE:



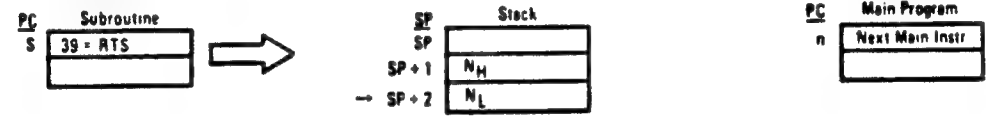


BSR, BRANCH TO SUBROUTINE:



JMP, JUMP.

RTS, RETURN FROM SUBROUTINE:



RTI, RETURN FROM INTERRUPT:

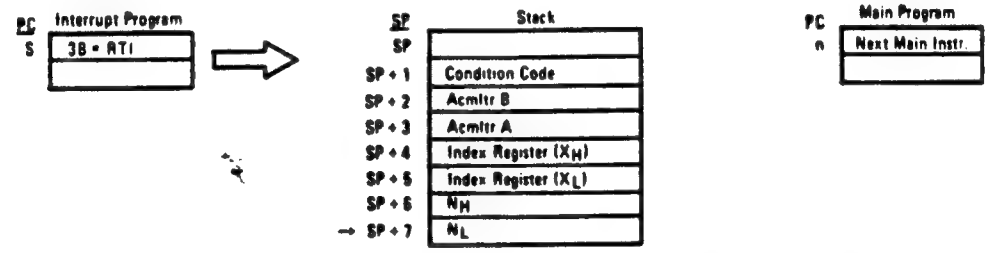


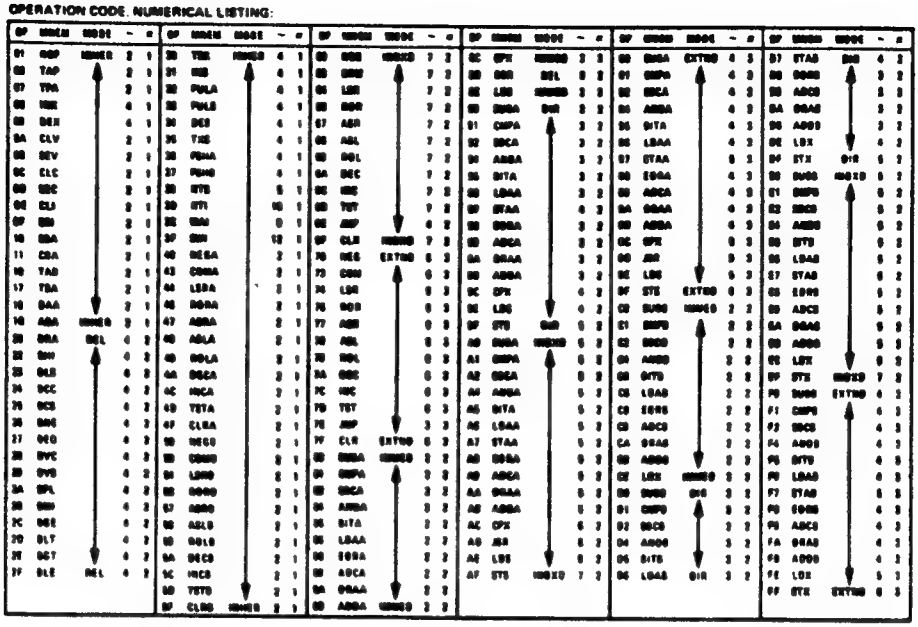
TABLE 6 – CONDITION CODE REGISTER MANIPULATION INSTRUCTIONS

OPERATIONS	MNEMONIC	IMPLIED OP	~	#	BOOLEAN OPERATION	COND. CODE REG. 5 H	4 I	3 N	2 Z	1 V	0 C
Clear Carry	CLC	0C	2	1	0→C	•	•	•	•	•	R
Clear Interrupt Mask	CLI	0E	2	1	0→I	•	R	•	•	•	•
Clear Overflow	CLV	0A	2	1	0→V	•	•	•	•	R	•
Set Carry	SEC	0D	2	1	1→C	•	•	•	•	•	S
Set Interrupt Mask	SEI	0F	2	1	1→I	•	S	•	•	•	•
Set Overflow	SEV	0B	2	1	1→V	•	•	•	•	S	•
Acmltr A→CCR	TAP	06	2	1	A→CCR	———	———	———	(12)	———	———
CCR→Acmltr A	TPA	07	2	1	CCR→A	•	•	•	•	•	•

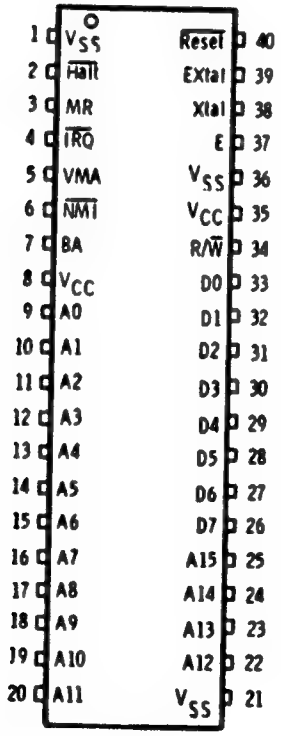
CONDITION CODE REGISTER NOTES: (Bit set if test is true and cleared otherwise)

1 (Bit V) Test: Result = 10000000?
2 (Bit C) Test: Result = 00000000?
3 (Bit C) Test: Decimal value of most significant BCD Character greater than nine? (Not cleared if previously set.)
4 (Bit V) Test: Operand = 10000000 prior to execution?
5 (Bit V) Test: Operand = 01111111 prior to execution?
6 (Bit V) Test: Set equal to result of N⊕C after shift has occurred
7 (Bit N) Test: Sign bit of most significant (MS) byte = 1?
8 (Bit V) Test: 2's complement overflow from subtraction of MS bytes?
9 (Bit N) Test: Result less than zero? (Bit 15 = 1)
10 (All) Load Condition Code Register from Stack. (See Special Operations)
11 (Bit I) Set when interrupt occurs. If previously set, a Non-Maskable Interrupt is required to exit the wait state.
12 (All) Set according to the contents of Accumulator A

OPERATION CODE NUMERICAL LISTING:



PIN ASSIGNMENT



6808

APPENDIX B

A TABLE OF SS MOTOR SELECT/SPEED/DIRECTION BYTES TO BE APPENDED TO THE C3,CC,D3, OR DC MOTOR MOVE COMMANDS. LIMITS ON ABSOLUTE POSITIONS XX FOR VARIOUS HERO MOTORS. (COMMAND FORMAT: C3(CC,D3,DC) SS XX).

In the motor command format C3(CC,D3,DC) SS XX, the C3 and CC command codes are associated with absolute motor positions XX, while D3 and DC are associated with relative motion (distances XX) of the motors. SS is calculated from the 8 bits mmmssDdd and XX from bits dddddddd. SS is associated with motor selected, speed desired, and (for the D3,DC commands) direction of motion. XX is associated with position or distance (absolute for C3,C3 or relative for D3,DC).

The rules are as follows (as demanded by HERO's interpreter monitor):

mmm = motor select = 000 (drive motor), 001 (extend/retract), 010 (arm(shoulder)), 011 (wrist rotate), 100 (wrist pivot), 101 (gripper), 110 (head rotate), 111 (steer). ss (speed bits) = 01(slow), 10(medium), 11(fast-not allowed for arm). D (1 or 0) selects the direction of motor travel (for the relative commands D3 and DC only - immaterial for the absolute position commands C3 and CC). The dd bits in the SS byte are ignored by the interpreter except in the case of the drive motor for HERO's base, in which case the dd bits in the SS byte for drive motor are appended to the dddddddd bits in the XX byte to give the distance drive motor is to traverse (hence increases the range of base (lateral) motion).

The above remarks allow us to calculate the SS bytes associated with each motor and with its speed we desire for absolute C3,CC or relative D3,DC motor move commands. The following table summarizes all the possibilities and makes a very handy reference for all the experiments we've done as well as for all others you contemplate. It saves us the headache of playing "tic-tac-toe" with the SS bits to come up with the appropriate byte in each case.

APPENDIX B

MOTOR	SPEED	SS BYTE FOR ABS. COMMAND (C3 or CC)	SS BYTE FOR REL. COMMAND (D3 or DC)	
			Forward: D=0	Reverse: D=1
DRIVE*	Slow	-	08-0B	0C-0F
	Medium	-	10-13	14-17
	Fast	-	18-1B	1C-1F
			Extend: D=0	Retract: D=1
ARM EXTEND OR RETRACT	Slow	28	28	2C
	Medium	30	30	34
	Fast	38	38	3C
			Up: D=0	Down: D=1
ARM (SHOULDER)	Slow	48	48	4C
	Medium	50	50	54
			Right: D=0	Left: D=1
WRIST ROTATE	Slow	68	68	6C
	Medium	70	70	74
	Fast	78	78	7C
			Down: D=0	Up: D=1
WRIST PIVOT	Slow	88	88	8C
	Medium	90	90	94
	Fast	98	98	9C
			Open: D=0	Close: D=1
GRIPPER	Slow	A8	A8	AC
	Medium	B0	B0	B4
	Fast	B8	B8	BC
			CW: D=0	CCW: D=1
HEAD	Slow	C8	C8	CC
	Medium	D0	D0	D4
	Fast	D8	D8	DC
			Right: D=0	Left: D=1
STEER	Slow	E8	E8	EC
	Medium	F0	F0	F4
	Fast	F8	F8	FC

*NOTE: There are no absolute positions for the motor drive. Hence use the relative motor move command codes D3,DC for motor drive. If, however, you do use the absolute commands C3 or CC for the motor drive case, then it will be interpreted as a relative command with XX defining the distance to be moved forward or back and with the D bit (0 or 1) in the SS byte appended to C3 or CC being "honored" (in this case only) as a valid direction bit. You can prove this to yourself with the following program:

```
0400    C3 08 10    Will drive HERO forward 10H units.
  03    C3 0C 10    Will drive HERO back 10H units.
  06    D3 08 10    Will drive HERO forward 10H units.
  09    D3 0C 10    Will drive HERO back 10H units.
  0C    3A          Return to monitor ("ready").
```

Note also that in the motor drive case we have given a range of SS bytes e.g. 08-0B, or 1C-1F, etc. This takes into account the possibility of increasing drive range through the two bits dd in the SS byte. They can assume the values 00,01,10, or 11 to increase drive range beyond that given by the 8 d bits in XX.

The absolute position range of motion possible with each motor on HERO, as well as each motor's initial absolute positions (relative only in the case of motor drive) are given below. These values may vary slightly from HERO to HERO.

MOTOR	INITIAL POSITION XX (ABS.)	ABS. RANGE OF POSITIONS XX
DRIVE	000	000-3FF
ARM EXTEND/RETRACT	00	00-98 (in to out)
ARM (SHOULDER)	00	00-86 (down to up)
WRIST ROTATE	4D*	00-93 (CCW to CW)
WRIST PIVOT	00	00-A5 (up to down)
GRIPPER	00	00-75 (closed to open)
HEAD	62**	00-BF (CCW to CW)
STEER	49**	00-93 (left to right turns)

* 4D makes gripper level with horizontal.
** 62 and 49 are straight ahead orientations of head and steer, respectively.